Mikko Saarinen

Sovellus IPv4- ja IPv6-osoitealueiden suunnitteluun

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

2.5.2016

Metropolia

Tekijä Otsikko Sivumäärä Aika	Mikko Saarinen Sovellus IPv4- ja IPv6-osoitealueiden suunnitteluun 44 sivua 2.5.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja	Yliopettaja Matti Puska

IP-osoitteiden hallinnan keskeisiä osa-alueita ovat IP-osoitesuunnitelman teko, aliverkotus sekä IP-osoitteiden ja DNS- ja DHCP-palveluiden ylläpito ja hallinta. Insinöörityössä perehdyttiin IPv4- ja IPv6-osoitealueiden suunnitteluun ja aliverkotukseen sekä toteutettiin komentokielinen sovellusohjelma, jolla voidaan käsitellä yritykselle tai organisaatiolle osoitettuja IP-osoitealueita. Sovellus auttaa verkkoasiantuntijaa IP-osoitesuunnitelman teossa, ehkäisee IP-osoitteisiin ja niiden muunnoksiin liittyviä virheitä ja helpottaa aliverkotuksen oppimista. Sovellusta voidaan myös käyttää aliverkotuksen harjoitusvälineenä ja IP-osoitteiden tunnistamiseen.

Insinöörityössä hyödynnettiin IP-osoitesuunnittelun perusperiaatteita, verkko- ja ohjelmistotekniikan kirjallisuutta sekä Go-ohjelmointiympäristön monipuolisia ominaisuuksia. Niiden pohjalta sovellus määriteltiin, suunniteltiin, ohjelmoitiin ja testattiin. Työn alussa laadittiin ensin projektisuunnitelma, jossa kuvattiin projektin resurssit, aikataulut, vaiheistus, riskit ja muutoshallinta. Projekti jaettiin useampaan eri vaiheeseen, joita vietiin läpi iteratiivisesti kahden, kolmen viikon jaksoissa. Kussakin jaksossa toteutettiin ja testattiin vaiheeseen liittyviä alatehtäviä.

Työn alkupuolella luotiin yksityiskohtainen vaatimusmäärittelyasiakirja, joka toimi myös projektissa toteutettavien alitehtävien lähteenä. Projektin suunnitteluvaiheissa laadittiin tarkennettu luokkakaavio, aliverkon eri vaiheiden tilakaavio ja kuva verkon binääripuusta. Sovelluskehityksessä ei noudatettu orjallisesti mitään tiettyä suunnittelufilosofiaa, ketterää tai perinteistä, vaan poimittiin yksittäisiä hyödyllisiä toimintatapoja, kuten versionhallinta, säännöllinen yksikkötestaus ja jatkuva integraatio. Työssä käytettiin Go-ohjelmointikieltä, graafista sovelluskehitysympäristöä Eclipseä ja Git-versionhallintaohjelmaa.

Työn tuloksena saatiin komentorivipohjainen, systemaattinen lähestymistapa IP-osoitteiden suunnitteluun ja muokkaukseen. Sovellus on ominaisuuksiltaan kehittyneempi kuin perinteiset, yksinkertaiset IP-osoitelaskurit, mutta samalla se on huomattavasti suppeampi kuin tarjolla olevat vapaan lähdekoodin tai kaupalliset IP-osoitteiden hallintajärjestelmät.

Avainsanat	IPAM, DDI, IP-aliverkotus, IP-osoitesuunnitelma

Author	Mikko Saarinen
Title	Application Program for IPv4 and IPv6 Address Planning
Number of Pages	44 pages
Date	May 2nd, 2016
Degree	Bachelor of Engineering
Degree Program	Information and Communications Technology
Specialization Option	Software Engineering
Instructor	Matti Puska, Principal Lecturer

The concept of IP address management consists of IP address planning, subnetting, maintenance of IPv4 and IPv6 addresses as well as DNS and DHCP services. The main objective of the final year project was to analyze, design, implement and test a command-line application program that can be used for managing the IP address space assigned to an organization.

The second objective was to get acquainted with project management and software development tools commonly used in software engineering. Initially, a project plan was written, describing the project schedule, the tasks, the risks and the change management. The project was divided into several phases, which were completed iteratively in two to three week intervals. The Go programming language, the Eclipse IDE and the Git version control system were used for the software development workflow.

The implemented application aids a network specialist in devising a hierarchical IP address plan, prevents errors and eases the learning of various subnetting techniques. While it is more advanced than plain IP calculators, it is substantially more limited in scope compared to any available open-source or proprietary IP address management system. In its present state, the program is already quite useful and can be extended with new features in the future.

Keywords	IPAM, DDI, subnetting, IP addressing plan

Sisällys

Lyhenteet

CIDR	Classless Inter-Domain Routing. Menetelmä kiinteiden IP-osoiteluokkien korvaamiseksi vaihtuvanpituisilla peitteillä reitityksessä. Reititys ilman IP-osoiteluokkia.
DDI	DNS, DHCP and IP Address Management. Vaihtoehtoinen termi IPAM:lle.
DNS	Domain Name System. Nimipalvelujärjestelmä. Järjestelmä, jolla muutetaan verkkotunnuksia IP-osoitteiksi ja IP-osoitteita verkkotunnuksiksi.
DHCP	Dynamic Host Configuration Protocol. Verkkoprotokolla, jonka avulla jaetaan IP-osoitteita ja verkon asetuksia laitteille.
IPAM	IP Address Management. IP-osoitteiden hallinta.
IPv4	Internet Protocol version 4. IP-protokollan versio 4.
IPv6	Internet Protocol version 6. IP-protokollan versio 6.
LIR	Local Internet Registry. Paikallinen IP-osoitteita asiakkaille myöntävä organisaatio ja RIR:n jäsen.
RIR	Regional Internet Registry. IP-osoitteita maantieteellisellä alueella rekisteröivä, hallinnoiva ja myöntävä organisaatio.
UML	Unified Modeling Language. Standardoitu kuvauskieli ohjelmistosuunnitteluun.
VLAN	Virtual LAN. Virtuaalilähiverkko, joka mahdollistaa fyysisen verkon jakamisen loogisiin osiin.

1 Johdanto

Insinöörityön aihe perustuu ajatukseen koulutusohjelman ohjelmistotekniikan pääopintojen ja tietoverkkojen perusopintojen yhdistämisestä mielekkäällä tavalla. Monialaisessa insinöörityössä tarkastellaan IPv4- ja IPv6-osoitealueiden suunnitteluperiaatteita teoriassa ja käytännössä ohjelmistototeutuksen kautta. Insinöörityöraportissa kuvataan IP-osoitteiden hallinnan yleisiä käsitteitä, perusperiaatteita, ongelmia ja toimintatapoja. Siinä tutustutaan myös työssä käytettyihin ohjelmistokehitysvälineisiin, kuten esimerkiksi Eclipse-ohjelmointiympäristöön ja Go-ohjelmointikieleen. IP-osoitteiden hallintaan liittyvää kirjallisuutta on melko vähän, ja aihetta käsitteleviä insinööri- tai opinnäytetöitä ei ole juurikaan tehty Suomessa.

Insinöörityön keskeisenä tavoitteena on määritellä, suunnitella, toteuttaa ja testata sovellusohjelma IPv4- ja IPv6-verkkojen ja -aliverkkojen suunnitteluun. Työ tehdään yksilötyönä ilman ulkopuolista toimeksiantajaa tai tilaajaa. Sovellusohjelmasta luodaan pilotti- tai esiversio, jossa toteutetaan IP-osoitesuunnittelun ja -aliverkotuksen perustoiminnallisuus, jota voidaan jatkossa laajentaa esimerkiksi graafisella käyttöliittymällä tai tietokantajärjestelmällä.

Sovellus auttaa uraansa aloittelevaa ja miksei kokeneempaakin verkkoasiantuntijaa IP-aliverkkojen muodostamisessa, nimeämisessä ja oikeellisuuden varmistamisessa, ja sitä voi käyttää myös harjoitusvälineenä. Koska IP-osoitesuunnitelman (IP address plan, IP addressing plan) laatimisessa on kohtalaisen helppo tehdä virheitä, jotka saattavat tulla esiin vasta tuotantovaiheessa, on järkevää käyttää sovellusohjelmaa virheiden ja ristiriitaisuuksien välttämiseksi etenkin vieraampien IPv6-osoitteiden osalta. Ohjelmaa käytetään komentokielen avulla merkkipohjaisesti, joskin alkuperäisen suunnitelman mukaan käyttöliittymän piti olla myös graafinen.

Insinöörityön toinen tärkeä tavoite on perehtyä projektinhallintaan, vaatimusmäärittelyyn, suunnitteluun, ohjelmointiin sekä yksikkötestaukseen käytännössä. Ohjelmistoprojekti rajataan ensisijaisesti IP-osoitteiden hallinnan osa-alueeseen, jossa tehdään yrityksen tai organisaation hierarkkinen aliverkkosuunnitelma. Projektin suunnittelu- ja toteutusvaiheissa hyödynnetään IP-osoitteiden hallintaa käsittelevää kirjallisuutta, suosituksia ja parhaita käytäntöjä sekä ohjaavan opettajan kommentteja ja palautetta.

2 IP-osoitteiden hallinnan perusteet

2.1 Tarve IP-osoitteiden ylläpidolle, seurannalle ja hallinnalle

IP-osoitteiden hallinta ja hallintajärjestelmät ovat yritysten ja organisaatioiden kriittisiä toimintoja. Jokainen verkon osana toimiva laite tarvitsee IP-osoitteen ja oikeat verkon asetukset, jotta tietoliikenneyhteys toisten verkon laitteiden kanssa olisi ylipäänsä mahdollista.

Nämä kriittiset toiminnot jäävät usein liian vähälle huomiolle, ja niihin ei aina suhtauduta riittävän vakavasti. Huolellisesti laadittu IP-osoitesuunnitelma on skaalautuva, dokumentoitu, keskitetty ja useamman kuin yhden henkilön käytettävissä. Yrityksen IP-osoiteavaruus on resurssi, jota on hallittava, ylläpidettävä ja seurattava kuten mitä tahansa verkonhallinnan osana olevaa resurssia.

Seuraavassa esitetään osin vaatimusmäärittelyasiakirjasta poimittuja ajatuksia ja perusteluja, miksi IP-aliverkkojen ja -osoitteiden suunnitteluun, ylläpitoon ja seurantaan pitäisi perehtyä huolellisesti:

- IP-osoitteiden hallinta on kriittinen resurssi.
- Organisaatiossa on saattanut esiintyä vikatilanteita IP-osoitteiden virheellisyyden tai kapasiteetin loppumisen takia.
- IP-osoitesuunnitelma ei ole riittävän joustava muutostilanteissa, kuten IPv6-käyttöönotossa.
- IP-osoitteiden hallintaan ei ole erillistä sovellusta, vaan ylläpito on tehty esimerkiksi taulukkolaskennalla.
- IP-osoitteiden ylläpito perustuu vain nimipalvelujärjestelmään (DNS, Domain Name System).
- Pelkkä yksinkertainen IP-laskuri ei palvele riittävästi IP-osoitesuunnitelman tekoa.
- Suunnittelua ei voi tehdä visuaalisesti graafisella työvälineellä.

- Monet apuvälineet eivät tue hierarkkisen suunnitelman tekoa.
- IPv6-osoitteiden tuki saattaa puuttua kokonaan.
- Sovelluksen avulla voidaan välttää suunnittelijan tekemiä virheitä.
- Aloitteleva tai kokematon verkkoinsinööri tarvitsee apuvälineen suunnittelun avuksi.
- Tuotteesta puuttuu tallennusmahdollisuus kokonaan.
- Tuote ei ole integroitu nimipalvelujärjestelmään tai IP-osoitteita jakaviin DHCP-palveluihin (Dynamic Host Configuration Protocol).
- Sovellus tehostaa ylläpitoa ja nopeuttaa muutoshallinnan ja vianselvityksen läpimenoaikoja.

Oma työkokemukseni järjestelmänhallinnan parissa tukee ja vahvistaa edellä esitettyjä havaintoja IP-osoitteiden hallinnasta ja siihen liittyvistä puutteista. Pienissä ja keskisuurissa yrityksissä ei välttämättä ole tunnistettu sen tärkeyttä, vaikka kyseessä on potentiaalisesti vakavia vikatilanteita ja tietoliikennekatkoja aiheuttava toiminto.

2.2 Käsitteet IPAM ja DDI

IP-osoitteiden hallinta eli IPAM (IP Address Management) on kriittinen yritysten ja internetpalveluntarjoajien verkonhallinnan toiminto, joka kattaa allokoidut IP-osoitealueet ja -osoitesuunnitelmat sekä niiden ylläpitoon tarvittavat verkon sovellukset ja palvelut, kuten DNS- ja DHCP-palvelut. IPAM koostuu kolmesta suuremmasta kokonaisuudesta, jotka ovat tiiviisti toisiinsa kytkettyjä. Nämä osa-alueet ovat 1) IP-osoitesuunnitelmat, aliverkotus, keskitetty hallinta ja seuranta 2) DHCP-palvelut 3) DNS-palvelut. Kyse on siis viime kädessä myös muutoksen- ja konfiguraationhallinnasta, missä yhden osa-alueen muutos vaikuttaa myös toisiin osa-alueisiin. (Rooney 2011: xi.)

DDI on lyhenne, joka muodostuu lyhenteistä DNS, DHCP ja IPAM. DDI-ratkaisuilla tuotetaan DNS-, DHCP- ja IP-osoitteiden hallintapalveluita, jotka alentavat kustannuksia, parantavat järjestelmähallintaa ja turvaavat kriittistä IT-infrastruktuuria. DDI on yhtenäinen, keskitetty hallintatapa kolmelle erilliselle verkon palvelulle. Sen avulla voi-

daan muun muassa määrittää DHCP-alueita (DHCP scope), tehdä IP-osoitteiden varauksia (reservation) ja tunnistaa hallinnan ulkopuolella olevia aliverkkoja. DDI:n tarkoituksena on parantaa verkon luotettavuutta, tuottaa seurantaraportteja, vähentää järjestelmänvalvojien työmäärää ja suunnitella verkon kapasiteettia. DDI-ratkaisut kattavat koko skaalan ilmaisista työkaluista suurten yritysten laitteistopohjaisiin ratkaisuihin. (Lerner & Canales 2015; Hubbard 2012.)

Termien IPAM ja DDI ero ei ole selkeä, ja siksi niitä käytetäänkin alan kirjallisuudessa vaihtelevasti. IPAM-asiantuntija Rooneyn (Rooney 2013a: 3) käsityksen mukaan termi IPAM kuvaa DDI:tä paremmin kokonaisvaltaista lähestymistapaa yksittäisten palvelujen sijaan. Joissakin DDI-tuotteissa painotetaan hieman enemmän DHCP- ja DNS-palveluiden merkitystä kriittisenä verkonhallinnan osana. 14.2.2016 tehdyn Google-haun perusteella hakusanat "DDI DNS DHCP" tuottavat 95 300 linkkiä, kun taas haku "IPAM DNS DHCP" antaa tulokseksi 179 000 linkkiä. Oma käsitykseni onkin, että IPAM on yleisempi ja mielestäni soveltuvampi termi IP-osoitteiden hallinnalle kuin DDI, jolla on useita muitakin merkityksiä tietotekniikan ulkopuolella (DDI 2016).

2.3 IP-osoitealueiden suunnittelu

Yrityksen tai yhteisön IP-osoitealueiden tehokas hyödyntäminen edellyttää hyvää suunnittelua, kapasiteetin kasvun ja muutosten mahdollisimman tarkkaa ennustamista. Se voi olla hyvinkin vaikeaa muuttuvassa maailmassa, jossa yritykset yhdistyvät, uusia toimipisteitä perustetaan ja vanhoja suljetaan. Myös muutokset käytettävissä sovelluksissa, kuten esimerkiksi VoIP-järjestelmän (Voice over IP) käyttöönotto, aiheuttavat paineita IP-osoitesuunnittelulle. IP-osoitesuunnittelun perustana voivat olla maantieteellinen sijainti, käyttäjien sovellusvaatimukset, Internet-yhteydet, tietoturva- ja auditointivaatimukset sekä järjestelmän- ja verkonhallinnan tarpeet. (Rooney 2011: 35–38.)

IP-osoitesuunnittelussa käytetään eri tasoja IP-osoitealueiden jakamiseen ja tunnistamiseen. Rooneyn teoksen esimerkissä käytetään neljää eri suunnittelutasoa, joista kolme on hierarkkisia ja yksi sovellukseen perustuva taso. Alueellisen hierarkian tasot ovat runkoverkko-, alue- ja toimipistetaso. Ylimmällä tasolla Rooney käyttää esimerkkinä 10.0.0.0/8-verkkoa, jonka hän jakaa ensiksi erilaisiin sovellusalueisiin. Alueet on nimetty seuraavasti: infrastruktuuri, VoIP-liikenne ja data-liikenne. Aliverkotuksen tuloksena saadaan yhtä suuret /12-aliverkot kullekin sovellusalueelle. Seuraavassa suunnit-

telun vaiheessa kukin sovellusalueen aliverkko pilkotaan optimaalisiin, erikokoisiin aliverkkoihin, joiden koko perustuu laite- ja käyttäjämääriin. Binäärilaskennan avulla yliverkko (supernet) pilkotaan toistuvasti pienemmiksi aliverkoiksi. Pilkkomisen jälkeen yliverkko ei enää ole käytettävissä, vaan ainoastaan puolitetut aliverkot. (Rooney 2011: 39–46.)

IP-osoitteiden allokointi vaatii loogisen suunnittelun lisäksi aktiivista seurantaa. Verkko, jonka käyttöaste on korkea, esimerkiksi 90 %, tarvitsee jatkuvaa seurantaa jopa useita kertoja päivässä, kun taas alhaisemman käyttöasteen verkko vaatii seurantaa vain muutaman kerran viikossa. (Rooney 2011: 38.)

Reitityksen suorituskyky, tehokkuus ja verkon luotettavuus ovat ominaisuuksia, joihin voidaan vaikuttaa hierarkkisen IP-osoitesuunnittelun avulla. Pienehkössä tai keskisuuressa verkossa, jossa on korkeintaan muutama sata aliverkkoa, reitityksen suorituskyky ei ole kovinkaan merkittävässä roolissa. Ongelma koskee lähinnä laajempia verkkoja, joiden tehokkuuteen voidaan vaikuttaa hyvällä IP-suunnittelulla. (Rooney 2013b.)

Seuraavassa kuvataan tarkemmin IP-osoitealueiden suunnitteluperiaatteita eri kriteerien perusteella.

Hierarkkinen jaottelu

Perinteinen tapa suunnitella keskisuuria ja suuria verkkoja on kolmiportainen malli, jossa runkoverkkotaso (core layer) yhdistetään aluetason keskuksiin (regional layer), joista liikenne puolestaan ohjataan loppukäyttäjien verkkoihin (access layer). Hierarkkisen IP-suunnitelman tulisi toteuttaa tätä mallia siten, että alemman tason aliverkot ositetaan vastaavan ylemmän tason osoitteista. Esimerkiksi organisaatiolle allokoitu /48-peitteellä oleva IPv6-osoite voidaan osittaa runkoverkon tarpeisiin /52-peitteellä. Kukin runkoverkon osoitealue voidaan jakaa /56-kokoisiin alueellisiin aliverkkoihin, jotka puolestaan voidaan pilkkoa esimerkiksi /60-kokoisiin loppukäyttäjien aliverkkoihin. Kukin reititystaso pystyy ohjaamaan liikennettä tehokkaasti alemmille tasoille yhdellä reititystaulun (routing table) yliverkon verkko-osoitteella. (Rooney 2013b.)

Hierarkkisessa IP-allokointimenetelmässä pyritään reititystaulujen pieneen kokoon, vähäiseen reitittimien väliseen reititystietojen vaihtoon sekä reittien tehokkaaseen yhdistämiseen (route aggregation) (Rooney 2011: 38).

Sovelluskohtainen tai käyttötapaan perustuva jaottelu

Käyttötapaan perustuva IP-verkkojen jaottelu voi kohdistua henkilöryhmään (oppilaat, opettajat, henkilöstöhallinta), laitetyyppiin (palvelimet, kytkimet) tai verkon osaan (julkinen, sisäinen, VPN) (Preparing an IPv6 Address Plan Manual 2013: 10).

IP-osoitesuunnittelussa voidaan käyttää esimerkiksi VoIP- tai dataliikenteelle varattuja sovelluskohtaisia IP-osoitealueita. Reititin tunnistaa sovelluksen tai käyttötavan saapuvan paketin lähdeosoitteen (source address) perusteella. Rooneyn esimerkissä määritetään loppukäyttäjän reitittimen parilliseen lukuun päättyvät verkko-osoitteet VoIP-käyttöön ja parittomaan päättyvät datakäyttöön vrt. 2001:db8:123:123**0**/64 ja 2001:db8:123:123**1**/64. Valitettavasti tämä yksinkertaiselta vaikuttava ratkaisu voi olla varsin hankala muutoshallinnan näkökulmasta. Laajassa verkossa pienikin VoIP-käytännön (policy) muutos kaikkiin parillisiin aliverkkoihin saattaa olla hyvin työlästä. (Rooney 2013b.)

Jos IP-verkko on jaoteltu aliverkkoihin käyttötavan mukaan, reititystaulun optimointi- ja tehokkuusnäkökulma ei ole keskeistä. Syynä tähän on se, että käyttötavan mukaista liikennettä välitetään tyypillisesti useisiin aliverkkoihin eri kohteisiin. Käyttötavan mukaisen jaottelun etu onkin yksinkertaisempi tietoturva ja pääsynhallinta, sillä useimmat palomuurit suodattavat liikennettä käyttötavan perusteella. (Preparing an IPv6 Address Plan Manual 2013: 9–12.)

Sijaintiin perustuva jaottelu

Sijainnin perusteella jaoteltavat IP-verkko-osoitteet ovat yleensä myös hierarkkisesti jaoteltuja. Verkon sijaintina voidaan käyttää esimerkiksi maantieteellistä aluetta, toimipistettä, rakennusta tai osastoa. Sijaintiin perustuvasta jaottelusta on se hyöty, että liikenteen ohjaaminen tiettyyn verkon sijaintiin vaatii vain yhden yliverkon määrityksen reititystauluun. Haittapuolena taas on se, että tietoturvapolitiikan ylläpito ja hallinta hankaloituu. (Preparing an IPv6 Address Plan Manual 2013: 10–13.)

VLAN-numerointiin perustuva jaottelu

VLAN-numeroiden mukaisessa jaottelussa virtuaalilähiverkon VLAN-numero sijoitetaan aliverkon tunnukseen joko suoraan heksalukuna, ulkomuodoltaan samanlaisena heksalukuna tai uudelleen koodattuna. VLAN-numeron ja IPv6-aliverkon tunnuksen yhdistämiseen on olemassa muutamia eri tapoja. 12-bittisen VLAN-tunnuksen lisäksi jää neljä bittiä muuhun käyttötarkoitukseen. Olemassa olevia VLAN-numeroita voidaan myös muuntaa toisenlaiseksi IPv6-osoitteeseen sijoitettaessa, jos VLAN-numerointi on esimerkiksi sijainti-käyttötapajärjestyksessä. (Preparing an IPv6 Address Plan Manual 2013: 15–17.)

Insinöörityön projektisuunnitelmassa tai vaatimusmäärittelyssä ei vaadita VLAN-numeroiden automaattista käsittelyä tai IP-aliverkko-osoitteiden muodostamista niiden avulla. VLAN-numeroinnin käyttö insinöörityösovelluksessa vaatii toistaiseksi verkko-asiantuntijan manuaalista työtä ja oman nimeämiskäytännön suunnittelua, joka voi osoittautua hankalaksi. Yksittäisen VLAN-tietokentän lisääminen IP-verkon tietorakenteeseen ei todennäköisesti olisi kovinkaan suuri työ.

2.4 IP-versiot osoitehallinnan näkökulmasta

IPv4- ja IPv6-osoitesuunnittelun periaatteet eroavat toisistaan huomattavasti. IPv4-osoitteiden allokointi perustuu ensisijaisesti tehokkuuteen, julkisten IP-osoitteiden saatavuuden rajoittamiseen ja mahdollisimman pieniin aliverkkoihin, kun taas IPv6:ssa tehokkuudella ei juurikaan ole merkitystä valtavan IP-osoiteavaruuden takia. IPv6-osoitesuunnitelmissa allokointi määräytyy sen sijaan sijainnin tai käyttötarkoituksen perusteella. (Preparing an IPv6 Address Plan Manual 2013: 3.)

Rooneyn (2011: 49) mukaan IPv4- ja IPv6-osoitteiden ulkoasu eroaa toisistaan, mutta molempien osoitetyyppien aliverkotusmenetelmä on periaatteiltaan sama. Eroja syntyy lähinnä siinä, että IPv4-osoitteessa muunnetaan desimaalilukuja binääriluvuksi, kun taas IPv6-osoitteessa muunnos tapahtuu heksaluvusta binärilukuun.

Siirtyminen IPv6-protokollan käyttöön etenee varsin maltillisesti ottaen huomioon, että IPv4-osoitteiden loppu (IPv4 depletion, IPv4 exhaustion) häämöttää jo. W3techs.com-sivuston kartoituksen mukaan vasta 6,1 % verkkopalvelujen tarjoajista käyttää IPv6-

protokollaa (w3techs.com 2016). RIPE NCC:n (Réseaux IP Européens Network Coordination Centre) laatiman tilaston mukaan 29.2.2016 IPv6-palveluita tarjoaa tasan 10 000 LIR:ä (Local Internet Registry), kun taas 3 182 LIR:ä ei tarjoa IPv6-palveluita (Nathalie 2016; RIPE NCC Statistics 2016).

2.4.1 IPv6-osoitealueiden suunnitteluperiaatteet

IPv6-osoitesuunnittelun keskeisimpiä periaatteita ja tavoitteita ovat reititystaulun pieni koko, reitityksen tehokkuus, helposti hallittava pääsynhallinta, sovellus- tai palvelukohtainen IP-suunnittelu, helpompi verkonhallinta ja IP-osoitteiden provisiointi, skaalautuvuus ja vianselvittely. (Rooney 2013b.)

Mitä etua saadaan huolellisella IPv6-osoitesuunnitelulla? Selkeitä hyötyjä ovat yksinkertaisempi hallinta pääsylistojen avulla (ACL, access control list), IP-osoitteiden tunnistettavuus käyttötavan tai sijainnin perusteella, verkon skaalautuvuus ja verkonhallinnan tehostuminen. (Preparing an IPv6 Address Plan Manual 2013: 3.)

Suunnittelu aloitetaan Rooneyn mukaan verkkoarkkitehtuurin IPv6-soveltuvuuden arvioinnilla. Tarkastelussa huomioidaan kaikki keskeiset tietojenkäsittelyn osa-alueet, kuten verkkoinfrastruktuuri, käytössä olevat laitteistot ja sovellukset. Työn etenemiseen tarvitaan mahdollisimman tarkka inventaari tai vähintään luettelo käytössä olevista resursseista ja dokumentaatiosta, myös IPv4:n osalta. Seuraavassa työvaiheessa hyödynnetään laite- ja ohjelmistotoimittajien antamia tietoja IPv6-yhteensopivuuden arviointiin. Jos tietoja ei ole saatavilla, IPv6-yhteensopivuus selvitetään testaamalla. Yhteen sopimattomat laitteet ja ohjelmistot vaihdetaan tai päivitetään uuteen versioon. Kartoituksen avulla voidaan tehdä IPv6-migraatiosuunnitelma, johon kuuluu muun muassa IPv6-osoitealueiden tilaaminen palveluntarjoajalta tai IP-osoitteita myöntävältä RIR-organisaatiolta (Regional Internet Registry), IPv6-osoitealueiden pilkkominen verkon topologian mukaisesti ja IPv6:n ja IPv4:n rinnakkaiskäyttö. (Rooney 2013b.)

IPv6-osoitteiden verkko-osa (prefix) on helpointa hahmottaa, kun sen pituus on neljällä jaollinen. Tällöin kukin 4 bitin jakso eli puolitavu (nibble) on kuvattavissa yhdellä heksadesimaaliluvulla. Aliverkkojen muodostaminen on mahdollista myös bitti kerrallaan, mutta silloin IPv6-osoitteen aliverkon raja osuu keskelle heksadesimaalilukua, mikä vaikeuttaa aliverkon hahmottamista. Tyypillisiä neljällä jaollisia verkko-osan pituuksia ovat /32, /48, /56, /60 ja /64. (Preparing an IPv6 Address Plan Manual 2013: 5.)

Insinöörityösovelluksen aliverkotustapa on oletusarvoisesti bitti kerrallaan, jolloin puolitavujen huomioiminen vaatii verkkoasiantuntijan omaa harkintaa. Puolitavujen mukainen aliverkotus on mahdollista käyttämällä esimerkiksi `sub prefix` -komentoa, joka pilkkoo verkon aliverkoiksi annetun verkko-osan pituuden mukaan. Tällöin käytetään neljällä jaollista verkko-osan pituutta.

Yksittäisen IPv6-osoitteen verkko-osan suositeltu pituus on /64, jotta esimerkiksi IP-osoitteiden automaattimääritykset (SLAAC, stateless address autoconfiguration) toimisivat. Yleinen palveluntarjoajien myöntämä /48-verkko sallii asiakkaan käyttää 16 bittiä omiin aliverkotustarpeisiin, mutta se voi olla liian suuri kotikäyttäjille. Tästä syystä myös IPv6-osoitteiden osalta pyritään osoittamaan sopivankokoisia verkkoja, kuten /56 tai /60, ja takaamaan tietty kasvuvara tulevaisuuden tarpeisiin. (Preparing an IPv6 Address Plan Manual 2013: 6, 19.)

Tärkeä elementti IP-osoitesuunnittelussa on visuaalinen tunnistaminen. Aivan kuten IPv4-osoitteetkin, myös IPv6-osoitteet voidaan suunnitella siten, että osoitteesta käy ilmi sijainti, sovellus, käyttötapa tai laitetyyppi. Visuaalinen tunnistaminen tapahtuu yleensä 128-bittisen IPv6-osoitteen prefiksin pituuden ja 64. bitin välisen alueen avulla. /48-peitteellä olevassa verkossa on siis käytettävissä 64 - 48 = 16 bittiä tunnistukseen, mikä tarkoittaa neljää heksadesimaalilukua. (Rooney 2013b.)

Kuvassa 1 on esitetty esimerkki IPv6-osoitteen jakamisesta aliverkkoihin sijainnin (location, L) ja käyttötavan (use type, T) perusteella. Kukin neljän bitin jakso vastaa yhtä heksalukua, joten osoite saa siis muodon 2001:db8:1234:**LTBB**::/64. Neljä bittiä mahdollistaa 16 eri sijaintia, joissa kussakin voi olla 16 eri käyttötapaa. Loppuosan kahdeksan bittiä sallii 256 erilaista sijainnin ja käyttötavan yhdistelmää. (Preparing an IPv6 Address Plan Manual 2013: 7–9.)

Kuva 1. IPv6-osoitteen jako aliverkkoihin ryhmittelyn avulla. L = sijainti, T = käyttötapa, B = vapaasti osoitettavissa. (Preparing an IPv6 Address Plan Manual 2013: 7.)

SurfNet suosittelee kuitenkin käyttötapa-sijainti-järjestystä IPv6-aliverkon tunnukselle (subnet id) (Preparing an IPv6 Address Plan Manual 2013: 10).

Poikkeus yleisestä RFC 4291:n (Hinden & Deering 2006: 7) /64-peitteellä olevan verkon periaatteesta on kahden laitteen välinen aliverkko, jossa käytetään Point-to-Point-linkkiä ja johon voidaan käyttää /127-aliverkkoa. Sen lisäksi peitettä /128 voidaan käyttää yksittäisessä liitännässä esimerkiksi reitittimen loopback-osoitteena. Muita mahdollisia suorien yhteyksien (Point-to-Point) verkko-osan pituuksia ovat /126, /120 ja /112. (Preparing an IPv6 Address Plan Manual 2013: 18.)

RIR:n tai palveluntarjoajan yksittäiselle asiakkaalle osoittaman IPv6-verkon ei enää nykysuosituksen mukaan tarvitse olla /48:n kokoinen. RFC 6177 kuitenkin suosittaa, että pienintä mahdollista /64-aliverkkoa ei käytetä edes kotikäyttäjille, vaan esimerkiksi /56-aliverkkoa, joka mahdollistaa laajentamisen ja kasvun tulevina vuosina. (Narten & Huston 2011: 5.)

Vaihtoehtoinen tapa IPv6-verkon pilkkomiselle on myös joustava jaottelu RFC 1219:n ja RFC 3531:n mukaisesti, missä aliverkko-osaan jätetään tyhjiä välejä tai bittejä tulevaisuuden kasvun varalle. Tyhjää tilaa jätetään esimerkiksi sijaintibittien ja käyttötavan tunnistavien bittien väliin. Näiden RFC-menetelmien käyttö vaatii myös hyvää osaamista binäärilukujen käsittelyssä. (Preparing an IPv6 Address Plan Manual 2013: 15.)

2.4.2 IPv4-osoitealueiden suunnitteluperiaatteet

IPv4-osoitteet jakautuvat kolmeen eri päätyyppiin, jotka ovat julkiset IP-osoitteet (public IP address), jotka koostuvat PA-osoitteista (Provider Aggregatable IP address) ja PI-osoitteista (Provider Independent IP address), yksityiset IP-osoitteet (private IP address) ja erikoiskäyttöön varatut IP-osoitteet (special-use IPv4 address). Internet-palveluntarjoajat voivat jakaa julkiset PA-osoitealueensa edelleen pienempiin osiin ja antaa niitä käyttöön toisille teleoperaattoreille tai asiakkaille. Palvelusopimuksen päättyessä IP-osoitteet palautuvat palveluntarjoajalle, jolloin asiakasverkon IP-osoitteet on numeroitava uudelleen. (IPv4 Address Allocation and Assignment Policies for the RIPE NCC Service Region 2015: 2–3, 9.)

IP-osoitteiden hallintajärjestelmissä tehtävään aliverkotukseen käytetään julkisia tai yksityisiä IPv4- ja IPv6-osoitteita (Rooney 2013a: 3–4). Erikoiskäyttöön varattuja IP-osoitteita ei käytetä kuin poikkeuksellisesti, kuten dokumentoinnissa tai ryhmälähetysosoitteissa (multicast address).

RFC 4632:n mukaan CIDR-notaatiossa (Classless Inter-Domain Routing) IP-osoite koostuu neljästä oktetista, vinoviivasta ja nollan ja 32:n välillä olevasta desimaaliluvusta. Vinoviivan jälkeinen luku kuvaa verkko-osan pituutta. IP-osoitteen verkko-osan jälkeiset, vähiten merkitsevät bitit ovat nollia. CIDR-tyyppiset IP-osoitteet helpottavat osoitealueiden määrittämistä tarpeen mukaan. Aikaisemmin käytössä olleiden luokkakohtaisten IP-osoitteiden sijaan voidaan käyttää mitä tahansa sopivaa verkko-osan pituutta. Käytännössä alle kahdeksan suuruisia verkko-osan pituuksia ei ole allokoitu. (Fuller & Li 2006: 4–7.)

IPv4-osoitealueiden suunnittelussa käytetään tyypillisesti optimaalista kapasiteettiin perustuvaa suunnittelua (best-fit allocation), jonka tavoitteena on säästää IPv4-osoiteavaruutta CIDR:n avulla (Rooney 2013b).

2.5 IP-osoitteiden hallintajärjestelmän toiminnot

Seuraavassa yleisen tason luettelossa kuvataan IP-osoitteiden hallintajärjestelmän käsitettä, joka koostuu kolmesta päätason toiminnallisuudesta. Verkon toimivuuden näkökulmasta ne kaikki luokitellaan kriittisiksi. (Rooney 2013a: 3–18; Rooney 2011: 306–346.)

IPAM-järjestelmän päätoiminnot ovat

1 IP-osoitealueiden suunnittelu, määrittely, ylläpito ja hallinta

- IP-osoitesuunnitelmat
- IP-aliverkotus
- IP-osoitteiden keskitetty hallinta

2 Dynaamisten IP-osoitepalveluiden hallinta DHCP:n avulla

- IP-osoitealueiden määritykset
- DHCP-palveluiden konfigurointi
- IP-osoitteiden käytön ja käyttöasteen seuranta

3 Nimipalveluiden hallinta DNS:n avulla

- staattisten tai dynaamisten DNS-nimien hallinta
- IP-osoitteiden ja toimialueiden (domain) nimeäminen
- DNS-palveluiden konfigurointi.

Nämä toiminnot ovat suurelta osin näkymättömiä verkon käyttäjille, mutta ilman niitä käyttäjät eivät pysty hyödyntämään mitään verkon resursseja. Tehokas ja virheettömästi toimiva IP-osoitteiden hallinta vaatii riittävää kapasiteettia ja huolellista DHCP- ja DNS-palveluiden määrittelyä. (Rooney 2013a: 3–4.)

Rooneyn mukaan verkonhallintakäytäntöjen soveltaminen IPAM:iin on aivan yhtä tärkeää kuin sen soveltaminen muihinkin kriittisiin verkon osiin ja laitteisiin. Yleisin verkonhallinnan menetelmä on FCAPS, mutta myös ITIL on käytössä yritysmaailmassa. FCAPS-termi muodostuu käsitteistä Fault Management, Configuration Management, Accounting Management, Performance Management ja Security Management. IPAM-järjestelmien kehityksen alkuvaiheessa IPAM miellettiin vain konfiguraation hallintavälineeksi, mutta se on myöhemmin laajentunut muihinkin FCAPS:n osa-alueisiin. (Rooney 2011: 305–307.)

Raskaan IPAM-järjestelmän sijaan voidaan aivan pienissä verkoissa käyttää muitakin ratkaisuja, kuten keskitetysti tallennettua taulukkolaskenta-asiakirjaa, Wiki-sivustoa, konfiguraation hallintajärjestelmää (CMDB Configuration Management Database) tai käänteisiä DNS-määrityksiä (Preparing an IPv6 Address Plan Manual 2013: 7). Olennaista on kuitenkin ymmärrys IPAM:n merkityksestä organisaation kriittisenä resurssina ja systemaattinen lähestymistapa IP-osoitesuunnitteluun.

2.5.1 IP-osoitteiden ja -osoitealueiden ylläpito ja hallinta

IP-osoitealueiden ylläpito ja hallinta on perusta verkon reititykselle ja muille IPAM-toiminnoille, kuten DHCP- ja DNS-palveluille (Rooney 2013a).

IPAM:n konfiguraationhallintatyö koostuu erilaisista hallinta- ja ylläpitotehtävistä. Keskeisiä tehtäviä ovat IP-osoitealueen, -aliverkon tai -osoitteen varaaminen käyttöön,

siirto toiseen verkkoon, poisto käytöstä, uudelleennumerointi (renumbering), pilkkominen aliverkkoihin ja yhdistäminen yliverkoksi. (Rooney 2011: 308–318.)

Eräs ongelma IPv6-osoitteiden hallinnassa on osoitteiden kirjoitusasun monimuotoisuus. IPv4-osoite on yksiselitteinen, pisteillä erotettu neljän desimaaliluvun joukko, mutta yksittäinen IPv6-osoite voidaan tallentaa useilla eri tavoilla, ja isojen tai pienten kirjainten käyttö ei ole vakioitua. Erilaiset kirjoitusasut hankaloittavat IPv6-osoitteen etsimistä järjestelmästä tai taulukkolaskennan taulukosta. Uuden IP-osoitteen yhteydessä tapahtuvan haun epäonnistuminen voi aiheuttaa päällekkäisiä IP-varauksia. RFC 5952:ssa listataan esimerkkejä yksittäisen IPv6-osoitteen eri kirjoitusasuista. RFC 5952 on suositus ja yritys vakioida IPv6-osoitteiden kirjoitusasu. (Kawamura & Kawashima 2010: 3–5.)

2.5.2 Aliverkotus ja verkkosuunnitelma

IP-aliverkkojen suunnittelu perustuu usein käyttäjien ja vierailijoiden määrään, käyttäjän tarvitsemien IP-osoitteiden keskiarvolukemaan, käyttöön tulevien IP-sovellusten määrään ja reititystarpeisiin. Suurten IP-osoitealueiden varaaminen ei aina ole mahdollista eritoten IPv4-osoitteiden osalta, joten järjestelmänvalvojien tehtäväksi jää optimaalisten osoitealueiden varaaminen toimipisteille. Laaja IPv6-osoiteavaruus ei vaadi yhtä tiukkaa rajausta, mutta selkeä, systemaattinen suunnittelu on silti hyvin tärkeää esimerkiksi tietoturvan ja pääsynhallinnan helpottamiseksi. (Rooney 2013a: 4.)

IPAM-järjestelmissä on erilaisia välineitä ja tapoja IP-osoitesuunnitelmien tekoon. Insinöörityössä pyrittiin selvittämään lyhyesti, minkälaisia ominaisuuksia IPAM-järjestelmissä ja IP-laskureissa on verkkosuunnitelmien laatimiseen ja aliverkotukseen. Ohjelmistojen ja apuohjelmien ominaisuuksia tarkasteltiin ohjekirjojen, esimerkkikuvien, teknisten raporttien (white paper) ja toimivien verkkosivujen avulla.

IPAM-järjestelmissä on lähes poikkeuksetta graafinen selaimella toimiva käyttöliittymä, jossa IP-verkko-osoitteet esitetään puumaisessa hierarkiassa. Joissakin pienemmissä ohjelmissa, jotka ovat lähinnä IP-laskureiksi luokiteltavia apuohjelmia, käytetään komentorivin komentoja, lippuja ja parametreja IP-osoitealueiden käsittelyyn.

Useimmiten uuden IP-verkon varauksen yhteydessä syötetään verkko-osoite, verkon peite tai pituus, verkon nimi ja mahdollinen tilatieto, VLAN-numero, verkon kuvaus tai

sijainti ja lisäksi mahdollisia kommentteja tai yhteystietoja. Uusi verkko voidaan ottaa suoraan tuotantokäyttöön sellaisenaan, siitä voidaan lohkaista aliverkkoja tai määritykset tehdään erikseen käsin.

2.5.3 IP-osoitteiden valvonta ja seuranta

IPAM-järjestelmissä IP-osoitteiden seuranta- ja valvontatehtävät ovat keskeisiä. On erittäin tärkeä tunnistaa organisaation verkkoihin kytketyt laitteet ja resurssit, sillä muutoshallinta käy hyvin hankalaksi, jos ei ole tarkalleen tiedossa, mitä IP-osoitteita tai -osoitealueita on aktiivisina ja käytössä.

IPAM-järjestelmien IP-osoitteiden seurannasta käytetään nimitystä "Accounting management". Se koostuu alatehtävistä, jotka ovat IP-inventaarin (IP inventory) ylläpito ja varmistaminen (inventory assurance), verkon kartoitus eri työkaluohjelmilla (network discovery), IP-inventaarin täsmäyttäminen (reconciliation) ja siihen läheisesti liittyvä IP-osoitteiden uudelleenkäyttö (IP address reclamation). IP-inventaari voidaan täsmäyttää vertaamalla sitä verkon kartoituksen tuottamiin tuloksiin sekä hyödyntämällä DHCP- ja DNS-palveluiden tuottamia raportteja. (Rooney 2011: 334–338.)

IP-osoitteiden valvonnassa huolehditaan siitä, että organisaatiossa noudatetaan IP-osoitesuunnitelmaa, seurataan konfiguraatiomuutoksia ja niiden valmistumista, etsitään IP-osoitteiden ristiriitaisuuksia ja vapaita resursseja uusiokäyttöön, edistetään verkon tietoturvaa ja noudatetaan mahdollisia palvelutasosopimuksia muutostöiden osalta (Rooney 2011: 334).

Suorituskyvyn hallinta (performance management) koostuu verkon palveluiden valvonnasta (services monitoring), IP-osoitealueiden kapasiteetin hallinnasta (address capacity management), auditoinnista (auditing) ja raportoinnista (reporting). Erittäin tärkeä seurannan kohde on verkkojen IP-kapasiteetti, joka perustuu IP-osoitesuunnitelman aikaisiin arvioihin laitemääristä ja kasvuennusteista. Liiketoiminnan muutokset, toimipisteiden muutot tai tilaajamäärien kasvu ovat esimerkkejä kasvavista IP-vaatimuksista, joten säännöllinen DHCP-valvonta onkin tärkeää. (Rooney 2011: 338–340.)

Jos aliverkkosuunnitelmassa on otettu käyttöön hyvin pieniä IPv4-osoitealueita, joissa ei juurikaan ole kasvuvaraa, seurantaa pitää tehdä päivittäin tai jopa useita kertoja päi-

vässä. IPv6-osoitteiden osalta kapasiteetin seuranta ei ole oleellista, koska missään /64-peitteellä olevassa aliverkossa ei ole puutetta IP-osoitteista. IPv6-verkkojen osalta suurempi ongelma onkin aktiivisten IP-osoitteiden kartoitus, jota ei voi tehdä lineaarisesti suuren osoitemäärän takia.

2.5.4 DNS-integraatio

Useat IPAM-tuotteet tukevat eri DNS-palvelinohjelmistoja, kuten ISC (Internet Systems Consortium) BIND:a ja Microsoft AD:ta. DNS-integraatiolla tarkoitetaan sitä, että DNS-palvelimen asetuksia voidaan ohjata suoraan IPAM-järjestelmästä ilman erillisiä ohjelmia tai laitteistoja. Tavoitteena on ylläpidon helppous keskitetyn konsolin avulla ja DNS-muutosten automaattinen päivitys kuhunkin DNS-palvelimeen.

Konfiguraationhallinnan näkökulmasta IPAM sisältää ainakin DNS-palvelimien konfiguroinnin, IP-osoitteiden ja toimialuenimien ylläpidon ja DNS-alueiden (zone) ja DNS-parametrien määritykset. Lisäksi siihen voi kuulua korkean luotettavuuden ja käytettävyyden määritykset (high availability) ja laitteisto- tai ohjelmistopohjaisten DNS-palvelimien konfigurointi. DNS-määrittelyjen täytyy myös nivoutua laadittuun IP-osoitesuunnitelmaan. (Rooney 2011: 307.)

2.5.5 DHCP-integraatio

IPAM-järjestelmissä voi myös olla tuki eri DHCP-palvelimille, kuten esimerkiksi Microsoft AD:lle, ISC DHCP:lle ja Cisco IOS DHCP:lle.

Aivan kuten DNS-integraationkin, myös DHCP-integraation tarkoituksena on helpottaa hallintatyötä ja automatisoida konfiguraatiomuutosten tekoa. IPAM-järjestelmissä IP-osoitteiden muutospyynnöt ohjataan useimmiten keskitetyn konsolin avulla kunkin DHCP-alueen vastaavalle palvelimelle. Integraatio vähentää tarvetta käyttää DHCP-palvelimien erillisiä graafisia tai komentorivipohjaisia työkaluja IP-osoitteiden muutoksissa, nopeuttaa muutoksia ja vähentää virheiden mahdollisuutta.

Konfiguraationhallinnan tehtäviin kuuluvat ainakin IP-osoitevaraukset (lease), DHCP-optiot (DHCP option), IP-osoitevarannot (IP address pool), ensisijaisten DHCP-palvelimien (primary DHCP server) ja vikasietopalvelinten (failover server) määritykset sekä tarvittaessa reitittimien DHCP-välitysagenttien (DHCP relay agent) määritykset.

Harvoissa IPAM-tuotteissa on suoraa tukea reitittimien muutoshallintaan. (Rooney 2011: 307.)

2.6 IPAM-järjestelmiä valmistavat yritykset ja yhteisöt

Omisteiset IPAM-järjestelmät

Markkinatutkimuksessa 2015 Gartner selvitti DDI-markkinoiden suuruutta ja suurimpia toimijoita. Tutkimuksen mukaan markkinoita hallitsee kymmenen yritystä. Viiden yrityksen tuotteet on suunnattu suurten yritysten laajoihin IP-osoitehallinnan tarpeisiin, ja viiden yrityksen tuotteet keskittyvät joihinkin DDI:n tai IPAM:n osa-alueisiin. DDI:n tarve on suoraan verrannollinen hallittavien verkkojen kokoon tai työntekijöiden määrään. Mitä suurempi yritys on kyseessä, sitä suurempi on myös keskitetyn DDI-järjestelmän tarve. (Lerner & Canales 2015.)

Gartnerin markkinatutkimuksen mukaan suurten yritysten markkinoilla toimivat BT Diamond, Alcatel-Lucent, BlueCat, EfficientIP ja Infoblox. Lisäksi seuraavat viisi yritystä edustavat hieman kohdennetumpia tai suppeampia IPAM-tuotteita: SolarWinds, Microsoft, Cisco Systems, Men & Mice ja FusionLayer. (Lerner et Canales 2015.) FusionLayer-nimen taustalla on aiemmin ollut Nixu Software Oy ja sen valmistama Nixu NameSurfer®-tuoteperhe.

Vapaan lähdekoodin IPAM-järjestelmät

Tarjolla on myös useita erilaisia vapaan lähdekoodin ohjelmistoja, kuten esimerkiksi GestióIP, IPplan, NOC, OpenIPAM, phpIPAM ja NIPAP (IP address management 2016). Tuotteiden kirjo on suuri ja vertailu on haastavaa, sillä sovellukset on toteutettu eri ohjelmointikielillä ja eri aikoina ja niiden laajuudessa on eroja. Lähes kaikissa tuotteissa on jo tuki IPv6-osoitteille, mutta vain alle puolessa on tuki VRF:lle (Virtual Routing and Forwarding). VRF:n tai VRF Liten avulla reititinverkoissa voidaan käyttää päällekkäisiä IP-osoitealueita ilman ristiriitoja.

Insinöörityöprojektin alkuvaiheessa vapaan lähdekoodin IPAM-tuotteista etsittiin lähinnä ideoita, miten komentokielisen tai graafisen käyttöliittymän aliverkotuksen voisi to-

teuttaa. Mielenkiintoista oli myös vilkaista joidenkin tuotteiden taustalla olevaa lähdekoodia (Python, Perl, JavaScript).

Laitteistopohjaiset IPAM-järjestelmät

Laajemmissa IPAM-järjestelmissä on lähes aina toteutettu sovelluspalvelimen ja DNS- ja DHCP-palvelimien välinen integraatio. Laitteistopohjainen IPAM-järjestelmä (IPAM appliance, hardware-based appliance) tarkoittaa IPAM-ratkaisua, jossa ohjelmisto on esiasennettu asiakkaalle toimitettavaan valmiiseen palvelinlaitteistoon. Tyypillisesti laitteistot ovat 1U- tai 2U-korkuisia räkkiin asennettavia palvelimia. Käyttövalmis palvelinlaitteisto voi sisältää IPAM-ohjelmiston lisäksi myös DNS- ja DHCP-palvelinohjelmistot.

Laitteistopohjaiset IPAM-järjestelmät on suunniteltu kriittisiin ja laajoihin korkean käytettävyyden verkkoympäristöihin. Palvelimissa on myös huomioitu käyttöjärjestelmän ja DNS- ja DHCP-palvelimien tietoturvatiukennukset (hardening) ja mahdollisuus keskitettyihin tietoturvapäivityksiin.

3 Sovelluksen suunnittelu- ja toteutusprojekti

3.1 Projektisuunnitelma

Insinöörityö oli resursseiltaan ja kooltaan pieni ohjelmistoprojekti, sillä projektilla oli yksi jäsen, ei lainkaan tiimiä ja asiakasnäkökulma oli hyvin suppea. Perinteisen saksalaisen suunnittelumallin, V-mallin, mukaan IT-projekti on kooltaan pieni, jos työmäärä henkilötyövuosina (man year) on 0,5 tai vähemmän tai projektin jäseniä on kaksi tai vähemmän (Vorgehensmodell V-Modell T.3 IT-Vorhabentypen 2004).

Pienestä koosta huolimatta työ vietiin läpi projektinomaisesti. Insinöörityöprojektissa laadittiin yleisiä projektilta vaadittavia asiakirjoja tai kuvauksia, kuten projektiehdotus, projektisuunnitelma, projektin ohjauspisteet, projektin muutoshallinta ja läpiviennin ja etenemisen seuranta (Haikala & Mikkonen 2011: 153–157). Projektisuunnitelmassa kuvattiin tärkeimmät osa-alueet, kuten projektin yleiskuvaus ja tavoite, projektin laajuus ja rajaukset, karkea vaiheistus, aikataulu ja tarkistuspisteet sekä riskit.

3.1.1 Aikataulu

Insinöörityön projektiaikataulu oli alusta alkaen suunniteltu tiukaksi. Projektin ensimmäinen vaihe, joka sisälsi muun muassa projektisuunnitelman laatimisen ja vaatimusmäärittelyn, oli ajoitettu alkavaksi 8.10.2015. Alkuperäisen aikataulusuunnitelman mukaan insinöörityön ja -raportin piti valmistua 1.3.2016 mennessä. Käytännössä projekti viivästyi alkuperäisestä aikataulusta noin neljä viikkoa, ja sen laajuutta jouduttiin supistamaan graafisen esitystavan osalta.

Aikataulusuunnittelun osittainen epäonnistuminen johtuu pääasiassa siitä, että projektin alustava aikataulu sovitettiin Metropolian insinööritöiden valmistumisaikatauluun ja se laadittiin siten, että valmistuminen keväällä olisi mahdollista. Aikataulun arviointi ei siis perustunut työmäärän arviointiin laskennallisen menetelmän, kuten COCOMO II:n tai toimintopisteanalyysin (function point analysis), avulla (Haikala & Mikkonen 2011:162). Arviointimenetelmien käyttö aikataulun laadintaan olisi ollut hankalaa, koska sovelluksen vaatimukset muuttuivat koko projektin ajan, funktioiden, tiedostojen tai luokkien määrä oli hyvin epäselvä varsinkin projektin alussa, aikaisempaa kokemusperäistä tietoa sovellusprojekteista ei ollut käytettävissä ja projektin vaikeusastetta oli hankala määrittää.

Projektinhallinnan työmäärä pyrittiin pitämään mahdollisimman kevyenä, koska oli odotettavissa, että toteutusvaihe, jossa opetellaan uutta ohjelmointikieltä ja -paradigmaa rinnakkain, vie suhteellisesti enemmän aikaa kuin vastaavan tutun ohjelmointikielen kanssa. Projektin työmäärää taas vähensi se, että aikaa ei kulunut tiimin ohjaukseen, viikoittaisiin tilannepalavereihin tai kehitystyötehtävien delegointiin.

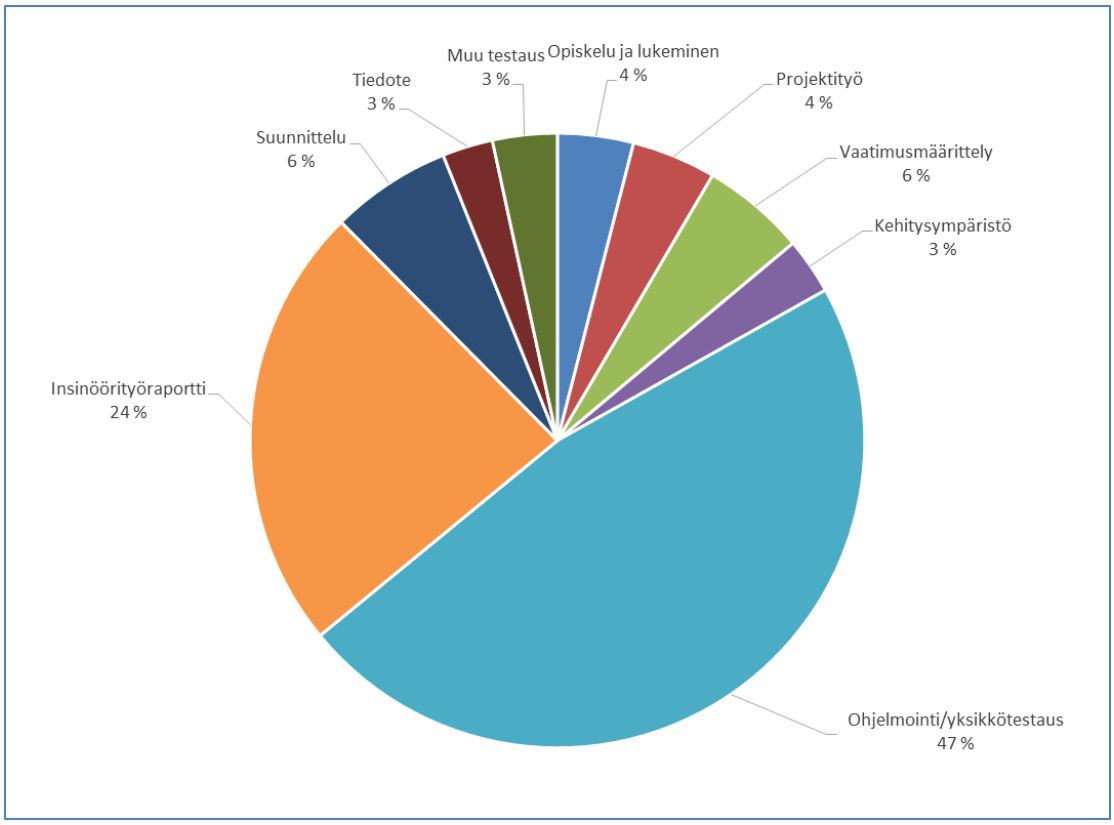
3.1.2 Etenemisen seuranta

Projektin etenemistä seurattiin suunnitelmatasolla kuukausittaisten ohjauspisteiden, viikkotiedotteen, käytettyjen työtuntien, vaihejaon ja lähdekoodirivien määrän (SLOC, source lines of code tai LOC, lines of code) avulla. Toteutusvaiheessa projektin etenemistä seurattiin vaatimusmäärittelyn alatehtävälistan, Git-versionhallintasovelluksen lokien ja yksinkertaisen TODO- ja virhelistan avulla.

Vaatimusmäärittelyn yksittäisiä vaatimuksia seurattiin hyvin yksinkertaisella tavalla merkitsemällä vaatimusasiakirjaan valmiit, keskeneräiset ja aloittamatta olevat tehtävät.

Projektin etenemisestä toimitettiin ohjaavalle opettajalle joka perjantai viikkotiedote, jossa kuvattiin kuluneen viikon tärkeimmät työtehtävät ja niiden arvioitu valmiusaste prosentteina. Viikkotiedotteen tarkoituksena oli lisätä ohjaavan opettajan ja insinöörityön tekijän yhteistyötä ja kommunikaatiota, etenkin kun kirjoittaja ei ollut aikaisemmin osallistunut ohjaavan opettajan kursseille. Toinen keskeinen viikkotiedotteen tehtävä oli varmistaa projektin sujuva, tasainen eteneminen, jolloin oli helpompi puuttua aikataulun pitämiseen tai eteen tulleisiin ongelmakohtiin.

Kuvassa 2 on havainnollistettu insinöörityöprojektiin käytettyjen työtuntien jakauma prosentteina. Noin neljäsosa eli 24 % projektin työajasta kului tämän raportin kirjoittamiseen ja 47 % ohjelmointiin, tekniseen suunnitteluun ja yksikkötestaukseen. Projektinhallintaan, joka koostuu projektityöstä ja projektin tiedottamisesta, käytettiin yhteensä vain noin 7 % kokonaisajasta.



Kuva 2. Insinöörityöprojektin työtuntien jakauma.

3.1.3 Riskit

Projektisuunnittelun yhteydessä pohdittiin mahdollisia aikatauluun vaikuttavia riskejä. Jokaisesta tunnistetusta riskistä kirjoitettiin lyhyt kuvaus, riskin todennäköisyys, riskin seuraukset, riskin ennaltaehkäisy ja korjaavat toimenpiteet (Haikala & Mikkonen, 2011: 164–166).

Projektin alkuperäinen aikataulu muuttui muutaman kerran projektin aikana. Syy aikataulun viivästymiseen selittyy osin toteutuneilla riskeillä. Riski uuden ohjelmointikielen tai -kirjaston viivästyneestä omaksumisesta toteutui ohjelmointivaiheen alkaessa ja aiheutti selkeää aikataulun viivästymistä. Toteutusvaiheen välineistöstä Go-ohjelmointiympäristö, Git-versionhallintasovellus ja Eclipse IDE kaikkine lisämoduuleineen olivat aivan uusia tuttavuuksia insinöörityön tekijälle. Uusien välineiden asentamiseen, testaamiseen ja opetteluun kului huomattavasti enemmän aikaa, kuin siihen oli varattu. Myös sairastumiseen liittyvät riskit ja taustamateriaalien läpikäyntiin liittyvät riskit aiheuttivat osaltaan viivettä.

3.1.4 Suunnitteluvälineet

Projektin eri vaiheiden suunnittelutyössä käytettiin tiiviisti Astah Professional 6.90 -mallinnussovellusta, jolla voidaan kuvata UML-standardin mukaisia kaavioita (Astah Professional Reference Manual 2013: 1). Astahilla laadittiin IP-osoitesovelluksen karkean tason luokkakaavio, verkko-osoitteen tilakaavio ja binääripuun (binary tree) hierarkiakuva. Sovellus toimi luotettavasti koko projektin ajan ja oli hyvä valinta, etenkin kun sitä oli käytetty aiemmin eri kursseilla. Ainoa todellinen haittapuoli Astahissa oli puuttuva tuki Go-ohjelmointikielelle (Astah Professional Reference Manual 2013: 406).

3.1.5 Toteutusvälineet

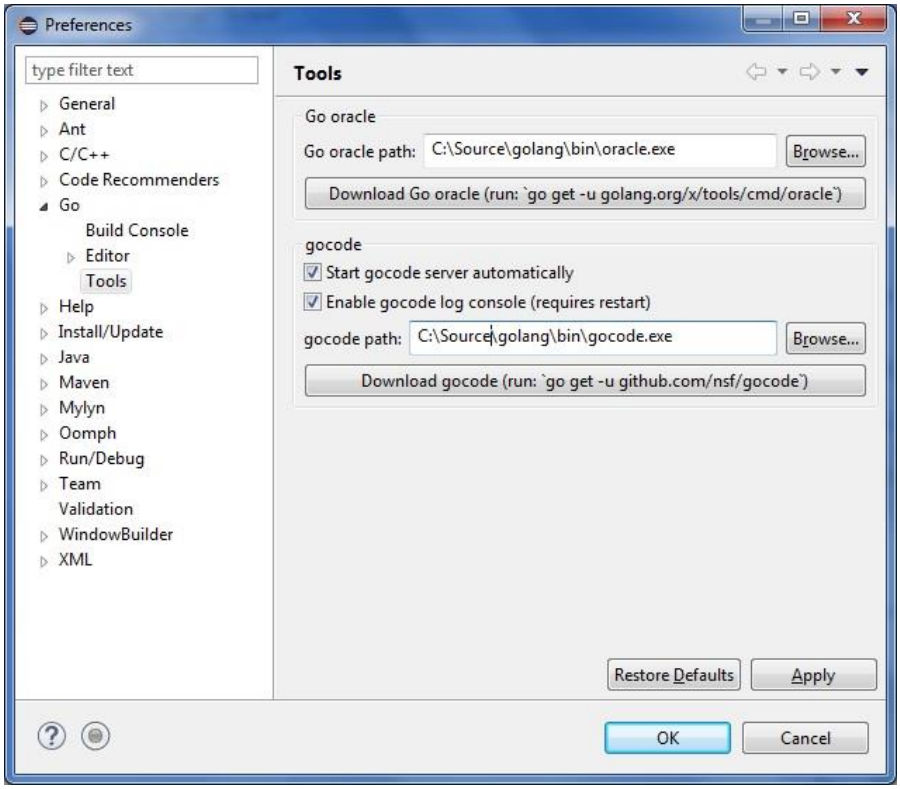
Projektin toteutusvaiheen välineistö oli melko tärkeässä roolissa, sillä uuden ohjelmointikielen, uuden versionhallintasovelluksen ja uusien ohjelmointiympäristöjen opettelu oli yksi insinöörityölle asetetuista tavoitteista, mutta samalla myös yksi riskeistä. Sovelluskehitykseen valittu Go vaikutti myös suunnittelu- ja toteutustapaan, vaikka näin ei periaatteessa pitäisikään olla. Gon olio-ominaisuudet poikkeavat perinteisistä olioperustaisista ohjelmointikielistä kuitenkin sen verran, että UML-mallinnus ei sujunut aivan totuttuun tapaan.

IP-suunnittelusovelluksen ohjelmointi tehtiin integroidulla ohjelmointiympäristöllä (Integrated Development Environment, IDE) Eclipse 4.5 Marsilla, joka vaatii toimiakseen Javan 32- tai 64-bittisen ajoympäristön (runtime environment). Projektin käyttöön asennettu versio oli nimeltään *Eclipse IDE for Java Developers 32-bit*, joka on ladattavissa verkko-osoitteesta https://eclipse.org/downloads/. Koska Eclipse IDE ei suoraan tue Go-ohjelmointikieltä, sovellukseen piti asentaa lisäkomponentteja ja apuohjelmia, kuten GoClipse, Gocode ja Go Oracle. Asennus oli osin automatisoitu, osin se piti tehdä käsin Go-perustyökaluja käyttäen. Taulukossa 1 on listattu Eclipsen integroituun ohjelmointiympäristöön asennetut apuohjelmat ja niiden kuvaukset.

Taulukko 1. Go-apuvälineet Eclipse IDE:ssä.

GoClipse	GoClipse on Go-lähdekoodin työkaluohjelma, jonka tarkoitus on syntaksin ja ohjelmointikielen elementtien tunnistus ja väritys (syntax highlighting), automaattinen sisennys (automatic indentation) ja sulkeiden täydennys (brace completion). GoClipsen avulla voi myös muokata Go-projektien ominaisuuksia tai käyttää siinä olevaa projektiavustajaa (project wizard). (Medeiros 2016.)
Gocode	Gocode on taustaprosessina (daemon) asiakas-palvelinarkkitehtuurin mukaisesti toimiva apuohjelma, joka integroidaan käytössä olevaan editoriin ja jolla voidaan automaattisesti täydentää Go-kielen käskyjä ja lausekkeita. Ohjelma käyttää MIT:n vapaan lähdekoodin lisenssiä. (Gocode 2016.)
Go oracle	Go oracle on työkalu lähdekoodin analysointiin. Se on kyselytyökalu, joka auttaa sovelluskehittäjää löytämään helposti esimerkiksi lähdekoodin määrityksiä, analysoimaan erilaisia lausekkeita ja niiden arvoja, tunnistamaan rajapintatoteutusten tietotyyppejä, etsimään kutsuvaa funktiota tai metodia sekä osoittimien kohteita. Go oracle -apuohjelma käynnistetään useimmiten lähdekoodieditorin tai integroidun kehitysympäristön sisältä. (Donovan 2015.)

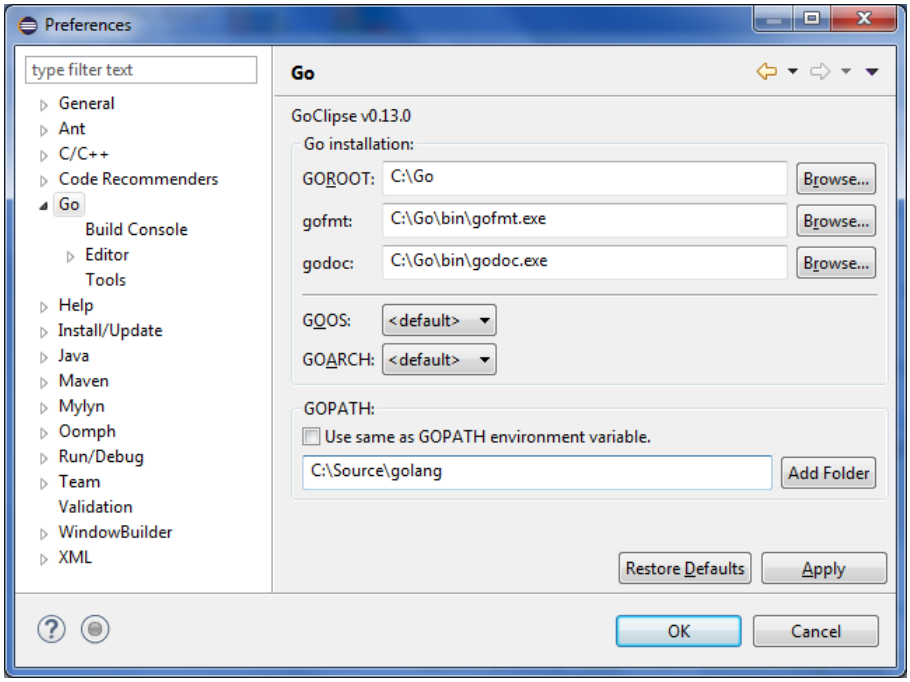
Kuvassa 3 on esitetty Go-apuohjelmien asetusnäyttö Eclipse IDE:ssä. Asetusnäytön kuvassa erottuu kaksi erillistä painiketta Go oracle- ja Gocode-apuohjelmien lataamiseen. Lataus tehdään taustalla Go-ohjelmointiympäristön komennolla `go get`, joka kopioi tarvittavat lähdekooditiedostot suoraan verkon palvelimelta ja kääntää ne käyttäjän työasemalla suoritettaviksi tiedostoiksi.



Kuva 3. Go oracle- ja Gocode-työkaluohjelmien määritykset Eclipse 4.5:ssä.

Kuvasta 4 nähdään gofmt- ja godoc-työkalujen sijainti työaseman tiedostohierarkiassa. Gofmt on apuohjelma kansion tai paketin lähdekooditiedostojen automaattiseen muotoiluun (Donovan & Kernighan 2015: 3). Se toimii erittäin hyvin käytännössä, mutta samalla pakottaa sovelluskehittäjän käyttämään yhtä tiettyä lähdekoodin muotoilutapaa. Sovelluskehityksen näkökulmasta muotoilun yhdenmukaistaminen parantaa lähdekoodin luettavuutta ja vähentää lähdekoodin muotoiluun liittyviä ristiriitoja.

Godoc on työkaluohjelma, jolla voidaan luoda Go-ohjelmien dokumentteja. Godoc jäsentää lähdekoodia ja tuottaa dokumentteja HTML- tai tekstimuodossa. Toisin kuin Javan JavaDoc, se ei vaadi erillisen syntaksin opettelua, vaan se käyttää suoraan funktion yläpuolella sijaitsevien kommenttirivien tekstejä dokumentoitiin. (Command godoc 2016; Gerrand 2011.)



Kuva 4. Go-kääntäjän ja -polun määritykset Eclipse 4.5:ssä.

Go-ohjelmointikielen tärkeimmät ympäristömuuttujat ovat GOROOT, joka viittaa asennuskansioon, GOPATH, joka viittaa lähdekoodihierarkian juurikansioon, ja GOBIN, joka viittaa suoritettavien ohjelmien asennuskansioon. Alla olevassa listauksessa näkyy ympäristömuuttujien sisältö Windows 7 -käyttöjärjestelmässä.

```
GOBIN=c:\source\golang\bin
GOPATH=c:\source\golang
GOROOT=C:\Go\
```

3.2 Go-ohjelmointikieli

Go sai alkunsa vuoden 2007 syyskuussa, ja se julkaistiin marraskuussa 2009. Suunnittelijat Rob Pike, Robert Griesemer ja Ken Thompson työskentelivät tuohon aikaan Googlella kielen parissa. (Donovan and Kernighan 2015: xi.)

Vapaan lähdekoodin Go-ohjelmointikielen ja -työkalujen suunnittelutavoitteita olivat ilmaisuvoima (expressiveness), nopeat käännökset ja hyvä suorituskyky sekä luotettavat, vakaat sovellukset. Gon taustalla vaikuttaa vahvasti C-kieli, mutta se on saanut vaikutteita myös monista muista ohjelmointikielistä, kuten ALGOL 60:sta, Pascalista, Modula 2:sta ja Oberonista sekä Rob Piken Squeak- ja Newsqueak-kielistä. Go sovel-

tuu erityisesti järjestelmä-, verkko- ja palvelinohjelmointiin. (Donovan & Kernighan 2015: xi–xiii.)

Go-ohjelmointikieltä pidetään yksinkertaisena tai helposti opittavana. Yksi sen suunnittelun lähtökohdista onkin ytimekkyys, joka käy ilmi Go-spesifikaation lyhyydestä. (Aimonetti 2015: 5.) Spesifikaatio on pdf-tiedostoksi tulostettuna vain noin 90 sivua. Insinöörityöprojektin perusteella Go ei kuitenkaan tunnu kovin yksinkertaiselta aloittelevalle ohjelmoijalle. Sen syntaksi poikkeaa muista olioperustaisista ohjelmointikielistä, kuten Javasta ja C++:sta, joihin perehdyttiin Metropolian kursseilla. Syntaksin omaksuminen ei kuitenkaan ole mielestäni aivan niin suuri kynnys kuin vakiokirjastojen ja kolmannen osapuolen kirjastojen omaksuminen, joka vie rutkasti enemmän aikaa. Ohjelmointikielen käytettävyys ja ilmaisuvoima perustuu paljon vakiokirjastojen laajuuteen ja tasoon eikä pelkästään sen peruselementteihin.

Gon ominaisuuksia

Go on yleiskieli (general-purpose language), joka soveltuu erityisesti järjestelmä- ja palvelinohjelmointiin (systems programming). Se on käännettävä, vahvasti tyypitetty ohjelmointikieli (strongly typed language), jonka erityispiirteisiin kuuluu muun muassa samanaikaisuus (concurrency), automaattinen roskienkeräys (garbage collection) ja suoritettavat, staattiset ohjelmatiedostot. (The Go Programming Language Specification 2015.)

Gon struct-tietorakennetta (data structure) voidaan pitää eräänlaisena olio-ohjelmointikielen luokkana, joka ei tue perintää, mutta tukee muodosteita (composition) (Aimonetti 2015: 21). Struct-rakenteen yksittäisiin kenttiin viitataan pistenotaatiolla (field selector) (The Go Programming Language Specification 2015).

Erot yleisiin ohjelmointikieliin

Taulukossa 2 kuvataan joitakin Gon kieliopin rakenteita ja vakiintuneita käytäntöjä, jotka erottavat sen muista yleisimmistä ohjelmointikielistä, kuten Java, C tai C++. Lista ei ole kattava vaan lähinnä suuntaa antava.

Taulukko 2. Gon erot yleisiin ohjelmointikieliin. (Effective Go - The Go Programming Language 2015; The Go Programming Language Specification 2015; Tkachenko 2015)

Gon ominaisuus	**Esimerkki**

Muuttujan tietotyyppi kirjoitetaan nimen jälkeen ja muuttujia voi ryhmitellä. Muuttuja alustetaan automaattisesti tietotyypin mukaan (0, nil, "" jne.).	`var (`
	`  laskuri int`
	`  Nimi  string`
	`)`

Taulukon koko on tietotyypin yhteydessä.	`var taulukko [5]int`

Sulkeita ei käytetä `if-`, `for-` tai `switch`-lausekkeissa.	`if x > 0 {`
	`  i++`
	`}`

Puolipistettä ei kirjoiteta lausekkeen loppuun, sillä Gon skanneri (lexical analyzer, lexer) lisää sen automaattisesti.	`if x > 0 {`
	`  a := Foo(x)`
	`}`

Go-ohjelmointikielessä ei ole `do-while-` tai `while`-lausekkeita vaan ainoastaan `for`-lauseke, jota käytetään myös tietorakenteiden slice ja map läpikäyntiin.	`for i := 0; i < 8; i++ {`
	`  fmt.Println("Laskuri", i)`
	`}`
	`for i, nimi := range nimilista {`
	`  fmt.Println(i, nimi)`
	`}`

Muuttujaan voi sijoittaa kaikkia tietotyyppejä ja myös funktioita ja sulkeumia (closure). Gossa voi käyttää etumerkittömiä kokonaislukuja.	`ok := func() { //kutsu ok()`
	`  fmt.Println("OK")`
	`}`
	`var b uint64`

Muuttujan määrittämiseen ja alustamiseen funktion sisällä käytetään usein lyhyttä `:=`-merkintää (short variable declaration).	`w := netaddr.wildcard()`
	`//luo ja alustaa oikeantyypisen`
	`//muuttujan w`

Julkiset kentät, metodit ja funktiot merkitään isolla alkukirjaimella. Go-suunnittelijat käyttävät näkyvyysmääreestä termiä "exported".	`type Asiakas struct {`
	`  id  int  //not exported`
	`  Name string //exported`
	`}`

Getter- ja setter-funktioille ei ole automaattista tukea.	`NetAddr  //getter-funktion nimi` `SetNetaddr  //setter-funktion nimi`

Funktio voi palauttaa useita paluuarvoja (multiple return values). Niitä käytetään usein virhetilanteiden käsittelyyn.	`tied, err := os.Open(tiednimi)`
	`if err != nil {`
	`  return err`
	`}`

Vakiintunut tapa on käyttää muuttujien ja funktioiden nimissä isoja ja pieniä kirjaimia (mixed caps, camel case), mutta ei alaviivaa.	`r := isPrivateAddress()  //ok` `r := is_Private_Address() //ei ok`
Gossa ei käytetä otsikkotiedostoja (header file), vaan jokainen lähdekooditiedosto on osa jotain tiettyä pakettia.	`package MyApp`
Gossa on automaattinen roskienkeruu ja muistinhallinta.	
Go tukee osoittimia, kuten C tai C++, mutta se ei tue osoitinlaskentaa (pointer arithmetic).	`var a *int`
Merkkijonot (string) ovat muuttumattomia luomisen jälkeen.	`var ok string = "Okay"`
Gosta puuttuu funktioiden kuormittaminen (function overloading).	

Gosta puuttuu poikkeuksien käsittely (exception handling) try-catch-tyyppisesti. Go käyttää sisäistä `error`-tietotyyppiä. Virheet tulisi käsitellä heti funktiokutsun jälkeen.	`x, err := foo(y)`
	`if err != nil {`
	`  log.Fatal(err)`
	`}`

3.3 Määrittely

3.3.1 Vaatimusmäärittely

Vaatimusmäärittely (requirements definition) oli insinöörityöprojektin kannalta hyvin keskeinen työvaihe. Usein vaatimusmäärittely tehdään yhdessä asiakkaan kanssa, jolloin projektin tehtäväksi jää vaatimusten toteuttaminen sovellukseksi. Koska insinöörityöprojektilla ei varsinaisesti ollut asiakasta tai tilaajaa, vaatimukset määriteltiin itsenäisesti ja ohjaavan opettajan tarkennusten perusteella. Toiminnallisessa määrittelyssä (functional specification) kuvattiin vaatimusten ohella myös ohjelmistosuunnittelun tehtäviä, ja se onkin siksi kohtuullisen laaja ja yksityiskohtainen.

Vaatimusdokumentissa (requirements specification, requirements document) kuvattiin

- sovellusalue
- perusteluja toteutukselle ja sovelluksella ratkottavia ongelmia
- vaatimusluettelo, nimeämiskäytäntö ja prioriteetit
- toiminnalliset vaatimukset (functional requirement)
- järjestelmävaatimukset (system requirement)
- kunkin vaatimuksen tarkemmat tiedot
- yksityiskohtaisia tietoja IPv4- ja IPv6-osoitealueista.

Vaatimusmäärittelystä jätettiin pois suurin osa ei-toiminnallisista vaatimuksista (non-functional requirements), kuten saatavuus (availability), skaalautuvuus (scalability), siirrettävyys (portability), tietoturva (security) ja käytettävyys (usability).

Järjestelmävaatimukset liittyivät lähinnä sovelluksen kehitysympäristön ominaisuuksiin, ajonaikaiseen ympäristöön ja käytettäviin selainversioihin.

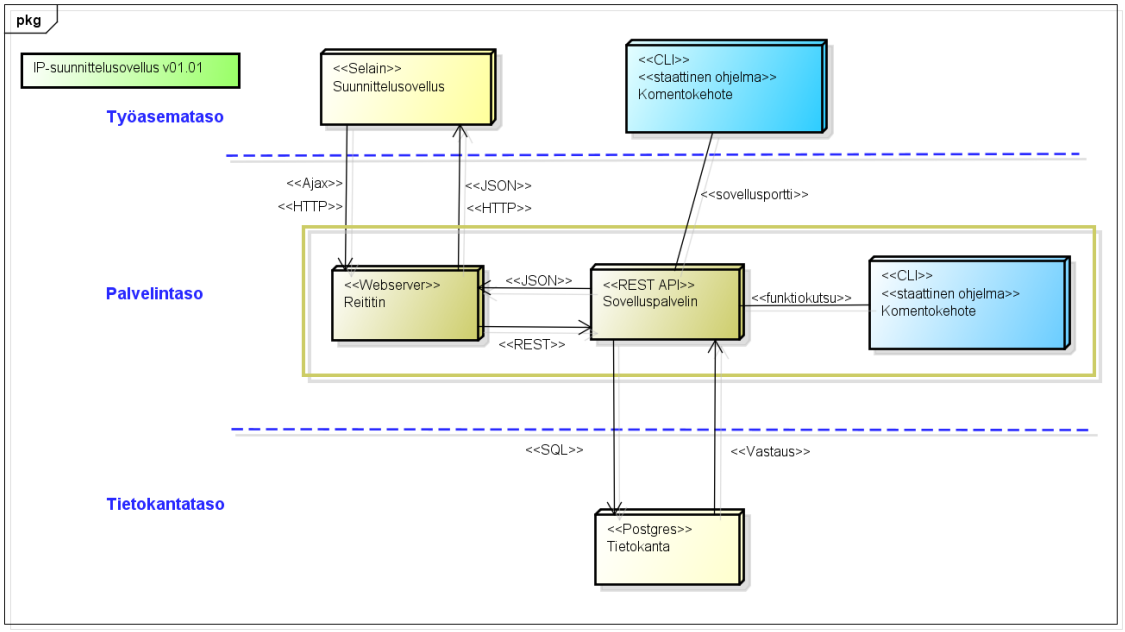
Vaatimusasiakirjan laadinta selkeytti ajattelua, pakotti perehtymään IPv4- ja IPv6-verkkojen ominaisuuksiin tarkemmin ja mahdollisti yhteisen kielen ja mielipiteenvaihdon ohjaavan opettajan kanssa. Kyse oli siis enemmän kuin pelkästä vaatimuslistasta, sillä

vaatimusmäärittely oli tärkeä väline aihealueen ymmärtämisessä, etenkin kun tekijän koulutuksen painopiste oli ohjelmoinnissa eikä verkkotekniikassa.

Vaatimusmäärittelytyön tuloksena saatu asiakirja oli hyödyllinen myös projektin suunnittelu- ja toteutusvaiheen yksittäisten tehtävien seurannassa ja toteutusjärjestyksen määrittelyssä. Vaatimusten muuttuessa projektin aikana vaatimusdokumenttia muokattiin iteratiivisesti varsinkin komentokielen syntaksin osalta.

3.3.2 Arkkitehtuurikuvaus

Sovelluksen arkkitehtuurikaaviossa (kuva 5) esitetään sovellusohjelmiston kokonaisarkkitehtuurin rakenne, mikäli kaikki suunnitelman viisi vaihetta toteutettaisiin. Kaavio heijastelee lähinnä tulevaisuuden tavoitetilaa, sillä on ilmeistä, että monen käyttäjän keskitetty tietojärjestelmä vaatii sekä palvelinohjelmiston että tietokantajärjestelmän. Kaaviosta käy myös ilmi REST-arkkitehtuurimallin (Representational State Transfer) ja selainkäyttöisen graafisen esitystavan tavoite.



Kuva 5. Suunnittelusovelluksen arkkitehtuurikaavio.

Insinöörityöprojektin määrittely rajattiin ensisijaisesti sinisellä väritettyihin laatikkoihin. Työssä keskityttiin IP-osoitteiden ja -osoitealueiden käsittelyrutiineihin, komentokielen määrittelyyn ja komentokehotteen rakenteeseen. Nykymuodossaan suunnitteluohjelma

soveltuu lähinnä yhden henkilön käyttöön, koska siinä ei vielä ole keskitettyä palvelinohjelmistoa, tietokantaa tai keskusmuistissa olevien tietorakenteiden samanaikaista käyttöä, joka vaatisi lukituksia. Sovellukseen jouduttiin kuitenkin toteuttamaan lisätyönä yksinkertainen IP-osoitealueiden tallennusmahdollisuus teksti- ja JSON-muodossa. Tallennusominaisuus oli välttämätön erityisesti komentokielen testauksen kannalta.

3.4 Suunnittelu

3.4.1 Sovelluksen luokkakaavio

UML 2.0 -standardi (Unified Modeling Language) määrittelee 13 erityyppistä suunnittelukaaviota, jotka on jaettu kolmeen eri ryhmään. Luokkakaavio (Class diagram) on osa UML-standardia ja kuuluu rakennekaavioiden (Structure diagram) ryhmään. (OMG UML 2015.)

Luokkakaavion käyttö on yleistä luokkien mallinnuksen ja oliopohjaisten ohjelmointikielten yhteydessä. IP-suunnittelusovelluksen yhteydessä luokkakaaviota käytettiin suunnitteluvaiheen lisäksi alkuvaiheen määrittelyssä lähinnä luokkien välisten suhteiden pohtimiseen vaatimusmäärittelystä poimittujen termien avulla. Suunnitteluvaiheessa luokkakaavio tarkentui kenttä- ja metoditasolle, joskin siinä esitetyt metodit ovat enimmäkseen julkisia (public). Kaavioon ei ole lisätty yksityisiä (private) metodeita tai suurta osaa apufunktioista, koska niiden määrän takia kaavio ei olisi kovin selkeä.

Insinöörityössä käytetyssä Astah Professional -suunnittelusovelluksessa on hyvä tuki luokkakaavioille, mutta valitettavasti se on tarkoitettu lähinnä sellaiseen oliosuunnitteluun, jossa hyödynnetään keskeisiä luokkamallinnuksen käsitteitä, kuten periytyminen (inheritance), yleistäminen (generalization) ja luokkahierarkia (class hierarchy). Astahissa on tuki ainoastaan Java-, C-, C++- ja C#-ohjelmointikielille. Sovelluksen toteutukseen käytetyssä Go-ohjelmointikielessä ei kuitenkaan ole tukea periytymiselle, eikä siinä käytetä tai tunneta yliluokkia (superclass) tai aliluokkia (derived class, extended class). Gossa ongelma ratkaistaan käyttämällä tietotyyppien kompositiota eli muodostetta (composition of data structs, composition), koostetta eli aggregaatiota (aggregation), upottamista (embedding) ja rajapintoja (interface).

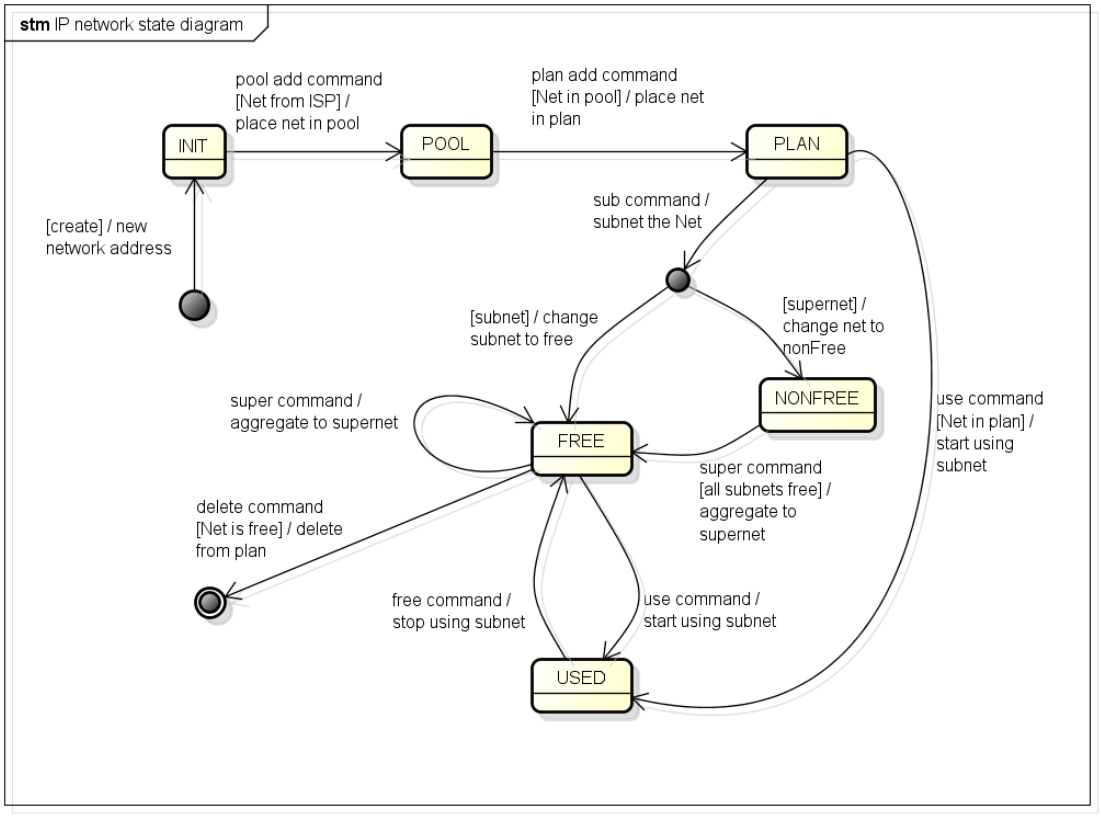
Kooste on tyhjällä vinoneliöllä ja assosiaatioviivalla kuvattava olion ja kokoelmaolion välinen assosiaatiosuhde tai yhteys. Koosteen kokoelmaoliossa viitataan jäsenolioon

esimerkiksi parametrin tai paikallisen muuttujan avulla. Jäsenolion elinkaari ei ole välttämättä sidottu kokoelmaolion elinkaareen. Muodoste on umpinaisella vinoneliöllä ja assosiaatioviivalla kuvattava yhteys, jossa isäntäolio omistaa jäsenolion isäntäolion koko elinkaaren ajan tuhoamiseen asti. Jäsenolio voi kuulua vain yhdelle isäntäoliolle kulloinkin. (Haikala & Mikkonen 2011: 90.)

Luokkakaavion hyödyntäminen insinöörityösovelluksen suunnittelussa oli hieman hankalaa, koska Astah ei tue suoraan Go-ohjelmointikieltä tai sen ominaisuuksia. Astahissa ei ole esimerkiksi mahdollista upottaa anonyymeja kenttiä (embedding anonymous fields), eikä siinä voi määrittää tietotyypin, sen kentän tai metodin näkyvyysmääreitä (visibility) ison tai pienen etukirjaimen avulla. Suunnittelutyötä vaikeutti myös se, että koulutuksessa on keskitytty perinteisiin olioperustaisiin C++- ja Java-kieliin sekä UML-suunnitteluun, mikä ei erityisesti auta Go-filosofian omaksumisessa.

3.4.2 IP-verkon tilakaavio

Projektin eri suunnitteluvaiheissa pohdittiin paljon yksittäisen verkko-osoitteen mahdollisia tiloja (state), tilasiirtymiä (transition) ja laukaisimia (trigger). Kuvassa 6 on esitetty suunnittelussa tunnistetut IP-verkon tai -aliverkon eri tilat ja käyttäjän antamilla komennoilla käynnistettävät tilasiirtymät. Kaavio laadittiin Astah Professional -sovelluksen avulla, ja se on tyypiltään UML-tilakaavio (state diagram).



Kuva 6. IP-verkon tai -aliverkon tilakaavio ja tilasiirtymät.

Yksittäisen IP-verkon elinkaari sovelluksessa alkaa verkko-olion luomisella. Gossa uusi instanssi voidaan luoda käyttämällä sisäistä `new`-funktiota, konstruktoria (constructor) tai komposiittiliteraalia (composite literal) (Effective Go - The Go Programming Language 2015). Luomisen jälkeen verkko-olio on olemassa sovellusprosessin keskusmuistissa tietorakenteena, jolloin se saa alkutilan INIT, mutta sitä ei vielä ole varsinaisesti liitetty järjestelmään. Komennolla **pool add <network address>** verkko-osoitteen oikeellisuus varmistetaan ennen sen viemistä varantoon (pool). Varannolla huolehditaan siitä, että kaikki RIR:n tai palveluntarjoajan yritykselle tai organisaatiolle osoittamat ylimmän tason verkko-osoitteet on tunnistettu ja rekisteröity järjestelmään ennen varsinaisen IP-aliverkkosuunnittelun aloittamista.

Varannon yksittäinen verkko-osoite voidaan siirtää osaksi yrityksen tai asiakasyrityksen IP-osoitesuunnitelmaa komennolla **plan add <network address>**, ja sille voidaan antaa kuvaava nimi **plan name <name>** -komennolla. Verkko-osoite saa tilaksi PLAN, joka tarkoittaa sitä, että verkkoa voitaisiin jo käyttää sellaisenaan tuotannossa.

Kun tietty verkko-osoite pilkotaan aliverkoiksi **sub**-komennolla, kukin aliverkko saa aina tilan NONFREE tai FREE. Samassa yhteydessä verkko-osoitteen oma tila muutetaan varatuksi NONFREE-tilaksi. Aliverkotuksessa luodaan aina vähintään kaksi aliverkkoa, jotka ovat yhtä suuria. Jako tehdään IPv4- tai IPv6-osoitteiden jakamisperiaatteen mukaisesti.

NONFREE-tilalla tarkoitetaan verkkoa tai aliverkkoa, jota ei voida sellaisenaan siirtää tuotantokäyttöön, koska sillä itsellään on toisia aliverkkoja. Kyse on siis eräänlaisesta varatusta verkosta, koosteverkosta (aggregated network address) tai yliverkosta, joka on CIDR-muotoinen. Varatun verkon alle on muodostettu aliverkkoja hierarkkisesti binääriaritmetiikan ja verkon peitteen pituuden (prefix length) muutoksen avulla. NONFREE-tilassa olevaa aliverkkoa ei voida enää pilkkoa pienemmiksi aliverkoiksi.

FREE-tilalla tarkoitetaan IP-osoitesuunnitelman yksittäisen verkon tai aliverkon alinta hierarkiatasoa. Tila FREE tarkoittaa vapaata kahdessa eri merkityksessä. Ensinnäkin FREE-aliverkko on vapaa, koska siitä voidaan muodostaa uusia pienempiä aliverkkoja, ja toiseksi se on vapaa, koska se voidaan ottaa tuotantokäyttöön **use**-komennolla. Tuotantokäytöllä tarkoitetaan tässä yhteydessä sitä, että aliverkko on nimetty, ja se voidaan määrittää reitittimeen, palomuuriin tai kytkimeen. USE-tilassa olevan aliverkon yhteyteen voidaan liittää yksittäisiä IP-osoitteita, kuten esimerkiksi varattuja, staattisia palvelimen IP-osoitteita.

3.4.3 Sovelluksen komentokieli

Projektisuunnitelman mukaan sovelluksen tekstipohjainen käyttöliittymä ja siihen tarvittava komentokieli suunnitellaan joko Go-apukirjastojen tai Ragel-jäsennintyökalun avulla. Sovelluksen komentokielen rakennekuvaus on osa vaatimusmäärittelyä, mutta se liittyy myös hyvin läheisesti tekniseen suunnitteluun. Ragel-työkalua oli myös tarkoitus käyttää tarvittaessa IP-osoitteiden ja IP-aliverkkojen oikeellisuuden tarkistukseen.

Ragel on ohjelmistokehitystyökalu säännöllisten lausekkeiden (regular expression) avulla tapahtuvaan jäsentämiseen (parsing). Sen avulla on mahdollista rakentaa hyvin monimutkaisia kielioppisääntöjä deklaratiivisesti ilman varsinaista ohjelmointia. Ragelin avulla kieliopista muokataan tilakoneita (state machine), jotka tallennetaan lähdekooditiedostoksi ja jotka voidaan integroida omaan sovellukseen. Ragelissa tuettuja ohjelmointikieliä ovat Go, C, C++, Objective-C, D, Java ja Ruby. (Thurston 2009: 1–5.)

Projektin alkuvaiheessa Ragelin tutkimiseen käytettiin aikaa muutamia viikkoja, mutta loppujen lopuksi se jätettiin toteutuksen ulkopuolelle. Syy tähän oli se, että Gossa on melko hyvät ominaisuudet IP-osoitteen oikeellisuuden tarkastamiseen, ja komentokielen syntaksi pystyttiin toteuttamaan riittävän hyvin kolmannen osapuolen kirjastolla.

Ragelin tilalle valittiin GitHub-palvelussa ylläpidettävä cli.go-komentorivikirjasto, jonka avulla suunnittelusovelluksen komennot ja alikomennot määriteltiin (GitHub codegangsta/cli 2016). Cli.go-kirjasto ei kuitenkaan soveltunut insinöörityösovelluksen tarpeisiin erityisen hyvin, sillä se on tarkoitettu lähinnä Unix-tyyppisten apuohjelmien toteutuksiin. Tyypillisesti Unix-työkaluohjelmat ovat komentorivillä erikseen käynnistettäviä prosesseja, jotka päättyvät komennon suorituksen jälkeen. Niissä käytetään usein myös komennon yhteydessä annettavia lippuja (flag). Insinöörityösovelluksessa tarvittiin kuitenkin toistuvasti annettavia komentoja, jotka suoritetaan aina saman ohjelmaprosessin sisällä. Sen lisäksi komentorivilipuille ei ollut mitään käyttöä.

Kuvassa 7 on nähtävissä **sub**-komennon määrittelyyn käytetyt cli.go-kirjaston parametrit. Kirjaston avulla on kohtuullisen helppo lisätä uusia komentoja ja alikomentoja tekstimuotoisena deklaratiivisesti. Action-kenttään voidaan lisätä käyttäjän oma funktiokutsu tai siihen voidaan upottaa koko funktio suoraan. Usage- ja Description-kenttien avulla tuotetaan automaattisesti komentojen avustustekstit (help text). Luonnollisesti komentojen suorittamiseen vaadittavat funktiot ja lausekkeet pitää ohjelmoida itse.

```
{
    Name: "sub",
    Aliases: []string{"b"},
    ArgsUsage:   "<count|size|prefix>",
    Usage:       "Subnet a network address",
    Description: "Subnet a network address by count, size or prefix",
    Subcommands: []cli.Command{
        {
            Name:      "count",
            ArgsUsage: "<count>",
            Usage:     "Split a network address into a number of equal-sized subnets",
            Action:    p.cmdSubCount,
        },
        {
            Name:      "size",
            ArgsUsage: "<count count ...>",
            Usage:     "Split a network address by host/node counts",
            Action:    p.cmdSubSize,
        },
        {
            Name:      "prefix",
            ArgsUsage: "<prefix>",
            Usage:     "Split a network address by prefix size",
            Action:    p.cmdSubPrefix,
        },
    },
},
```

Kuva 7. Sub-komennon määrittely cli.go-kirjaston avulla.

3.5 Toteutus

Toteutusvaiheen osuus työtunneissa mitattuna oli lähes puolet projektin kokonaistyömäärästä. Projektin toteutusvaiheeseen kuuluivat tekninen suunnittelu, ohjelmointi, myös paperilla tapahtuva funktioiden ja metodien tarkempi suunnittelu sekä yksikkötestaus. Yksikkötestaus oli niin olennainen osa ohjelmointia, ettei sen osuutta voinut laskea erikseen. Usein muutaman lähdekoodirivin editoinnin jälkeen muokattiin heti myös vastaavaa yksikkötestiä, jolloin voitiin varmistua muutoksen toimivuudesta.

3.5.1 Verkko-osoitehierarkian tietorakenne

IP-aliverkotuksessa verkko-osoite jaetaan aina kahteen samankokoiseen puolikkaaseen, mikä vastaa hyvin binääripuun käsitettä. Sovelluksen verkko-osoitehierarkian tietorakenteeksi valittiin keskusmuistissa toteutettu binääripuu, jonka juuri (root) on itse verkko-osoite ja jonka solmut (node) ja lehdet (leaf) ovat aliverkkoja. Binääripuun solmuun tallennetaan kaikki tarpeelliset tiedot yksittäisestä verkko-osoitteesta, kuten muun muassa IP-osoite kahtena 64-bittisenä kokonaislukuna, verkko-osan pituus (prefix length) ja verkon nimi.

Vaihtoehtoinen tietorakenne IP-osoitteiden käsittelyyn olisi ollut binäärinen trie (binary trie, prefix tree), joka soveltuu hyvin IP-osoitteen verkko-osan tallentamiseen ja tunnistamiseen reitittimissä. Trie-tietorakenne perustuu verkko-osan yksittäisten bittien muodostaman polun käsittelyyn. Trie-tietorakenteessa verkko-osaa tai avainta (key) ei tallenneta solmuun, sillä verkko-osan bittipolku toimii avaimena. (Deepankar & Karthikeyan 2007: 495–519.)

Toteutusvaiheen hankalimpia osuuksia oli binääripuuhun tallennetun verkko-osoitteen käsittely rekursiivisesti. Koska GitHubista tai Go-vakiokirjastoista ei löytynyt täsmälleen sopivaa valmiskomponenttia, jossa olisi huomioitu IP-verkon numerointi ja erilaisia tietoalkioita tai kenttiä, projektissa toteutettiin räätälöity käsittelykirjasto. Koska aikaisempaa kokemusta vastaavista puualgoritmeista ei juurikaan ollut, toteutuksesta tuli melko työläs ja monivaiheinen. Binääripuun käsittelyä esijärjestyksessä (pre-order) tehtiin useilla tavoilla toteutustyön eri vaiheissa. Sovelluksessa käytettyjä binääripuun läpikäyntimenetelmiä olivat 1) jokaisen solmun tietojen tulostus näytölle 2) laskurin arvon tulostus jokaisen tietorivin alkuun 3) yksittäisen verkko-osoitteen etsiminen ja totuusar-

von palautus 4) verkko-osoitteiden etsiminen ID-tunnuksella tai nimellä ja 5) haluttujen solmujen muistiosoittimien (pointer) tallennus erilliselle listalle jälkikäsittelyä varten.

Ensimmäiset binääripuun käsittelyversiot eivät mahdollistaneet myöhempien vaiheiden monipuolisempia hakutapoja tai tiedon palauttamista kutsuvalle metodille, joten algoritmeja piti kehittää ja muunnella koko toteutusvaiheen ajan. Viimeisessä kehitysversiossa käytettiin apuna muokattua Vierailija-suunnittelumallia (Visitor design pattern). Vierailija-suunnittelumallin avulla kussakin oliossa voidaan suorittaa tietty rajapintafunktio ja kerätä sen avulla tietoa oliosta ja sen kentistä. Tätä suunnittelumallia tarvitaan erityisesti silloin, kun on tarve suorittaa erilaisia algoritmeja oliorakenteessa. Oliot voivat olla myös eri luokkien ilmentymiä, jos ne toteuttavat tarvittavat rajapinnat. (Kerievsky 2005: 320–328.)

3.5.2 Toteutuksen ongelmakohtia

Maailmalla hyvin tunnetun ohjelmistokehittäjän ja kirjailijan Martin Fowlerin (2003) mukaan teknistä velkaa (technical debt) syntyy, kun sovellukseen lisätään toimintoja nopeaan tahtiin huolehtimatta selkeästä ja ylläpidettävästä ohjelmarakenteesta. Tekninen velka on Fowlerin mukaan yhdysvaltalaisen ohjelmoijan Ward Cunninghamin kehittämä termi, jolla kuvataan juuri nopean kehitystyön ja huonosti hallitun ohjelmistorakenteen ristiriitaa. Tekninen velka aiheuttaa väistämättä lisätyötä tulevaisuudessa, mutta velan vaikutuksia voidaan vähentää sovelluksen refaktoroinnilla. Velan kasvattaminen on sinänsä aivan hyväksyttävää esimerkiksi liiketoiminnan edistämiseksi projektin takarajan lähestyessä. Toisaalta tekninen velka voi muuttua isoksi ongelmaksi, jos se kasvaa liian suureksi ja alentaa tiimin tuottavuutta.

Kehitystyön eri vaiheiden seurauksena insinöörityösovelluksen ohjelmakirjastoon kertyi vähitellen useita eri periaatteilla toimivia funktioita ja metodeja, joista ongelmallisimpia olivat binääripuun eri käsittelyversiot. Kirjastoon muodostui siis teknistä velkaa. Koska kehitystyössä piti edetä nopeasti tiukan aikataulun takia, jo toteutettujen funktioiden refaktorointiin ei ollut aina riittävästi aikaa. Suunnittelun puutteiden korjaaminen myöhemmässä vaiheessa osoittautui yllättävän hankalaksi. Koodin muokkaus yhtenäisemmäksi ja päällekkäisyyksien poisto osoittautui vaativaksi etenkin testien puuttuessa.

Tekninen velka -käsitteen ohella käytetään myös termiä *under-engineering*. Kerievskyn (2005: 3–4.) mukaan sillä tarkoitetaan ohjelmistojen huonoa suunnittelua, joka johtuu kiireestä, refaktoroinnin puutteesta, ohjelmistosuunnittelun osaamisen heikkoudesta, liian nopeasta toimintojen lisäyksestä tai liian monista rinnakkaisista projekteista. Pidempään jatkettuna huono suunnittelu johtaa hidastuvaan sovelluskehitystahtiin, suuriin ylläpidollisiin haasteisiin ja jopa yrityksen kilpailukykyongelmiin.

Toteutustyö eteni aluksi melko vaivalloisesti, koska Go-ohjelmointikieli, Eclipse-ohjelmointiympäristö ja Git-versionhallintasovellus eivät olleet tuttuja entuudestaan. Koska aliverkotukseen tarvittavat IP-ohjelmakirjastot olivat alussa keskeneräisiä, protoilu ja integraatiotestaus oli hankalaa ja hidasta. Työ helpottui selvästi toteutusvaiheen loppupuolella, sillä uusien komentojen ohjelmointi aikaisemmin toteutettujen komponenttien ja funktioiden yhdistelyllä sujui ajoittain jouheasti.

3.6 Testaus

Projektisuunnitelman mukaan sovellusta testataan toteutusvaiheessa jatkuvasti yksikkötestauksen periaatteiden mukaan. Tavoitteena oli laatia jokaiselle itse toteutetulle moduulille, funktiolle ja metodille testi, jolla testattaisiin perustoiminnallisuutta sekä oikeilla että virheellisillä testiaineistoilla. Projektisuunnitelmassa ei erikseen mainita integraatiotestausta tai järjestelmätestausta, eikä erillistä testaussuunnitelma-asiakirjaa katsottu tarpeelliseksi näin pienessä projektissa.

Toteutuksen edetessä kävi hyvin selkeästi ilmi, että testejä tarvittiin sovelluksen oikeellisuuden toteamiseksi. Lähdekoodin refaktorointi oli myös helpompaa ja luotettavampaa, kun testit olivat olemassa. Useamman kerran lähdekoodiin tehty muutos tai virheenkorjaus aiheutti testien epäonnistumisen.

Yksikkötestaus

Käytännössä yksikkötestaus (unit level testing) painottui IP-ohjelmakirjaston metodien ja keskeisimpien tietorakenteiden testaamiseen. Etenkin toteutuksen alkuvaiheessa lähes jokaiselle IP-lähdekoodikirjaston funktiolle tai metodille luotiin testifunktio ja tarvittavat testiaineistot. Testauksen kattavuus IP-lähdekoodikirjaston osalta oli 1,5 kuukauden toteutustyön jälkeen 87 % ja 2,5 kuukauden jälkeen hieman alhaisempi, 82 %.

Toteutusvaiheessa sovelluksen koontiversiota (build) testattiin tiiviisti, usein jopa muutaman lähdekoodirivin kirjoittamisen jälkeen. Sovelluksen kehitys- ja testaustapa muistutti toisinaan TDD-ajattelua (Test Driven Development), sillä yksittäisen funktion tai metodin toteutus aloitettiin usein testin laatimisella. On kuitenkin todettava, että missään työn vaiheessa ei saavutettu TDD:n vaatimaa yksikkötestien kattavuutta. Projektin myöhemmässä vaiheessa uusien toimintojen ja vaatimusten toteutustyö ei edennyt riittävän ripeästi, jolloin testauksen määrä jäi selkeästi pienemmäksi kuin alkuvaiheessa.

Integraatiotestaus

Integraatiotestaus tuli varsinaisesti ajankohtaiseksi vasta toteutuksen toisessa vaiheessa, jossa käytettiin ulkopuolista vapaan lähdekoodin cli.go-komentorivikirjastoa. Projektin ykkösvaiheen IP- ja tietorakennekirjastojen sekä komentorivikirjaston yhdistämisessä tuli heti esiin useita selkeitä puutteita, virhetilanteita ja suoranaisia ohjelman kaatumisia. Suurin osa virheistä johtui puutteellisista syötteiden tarkistuksista, tyhjistä osoittimista (nil pointer) ja puuttuvista virheilmoituksista.

Integraatiotestaus keskittyi sovelluksen IP- ja komentokielikirjaston väliseen testaukseen. Suurin osa integraatiotesteistä tehtiin käsin syöttämällä komentoja suoraan ohjelmaan tai kopioimalla valmiita komentosarjoja ohjelman syötteiksi. Osasyy vähäiseen integraatiotestaukseen oli myös se, että tietokantajärjestelmän tai HTTP-palvelimen suunnittelu ja toteutus eivät kuuluneet projektin laajuuteen.

Suorituskykytestaus

Suorituskykytestien mukaan 1 024 aliverkon luonti keskusmuistiin binääripuuhun tapahtuu viiveettä, ja 2 048 aliverkon luonti kestää noin kaksi sekuntia. Suurten tai hyvin suurten aliverkkomäärien luonti yhdellä komennolla tapahtuu sovelluksessa aivan liian hitaasti, sillä esimerkiksi 32 767 aliverkon luonti kesti jo 6 minuuttia! Voimakas hidastuminen aliverkkomäärän kasvaessa johtunee siitä, että keskusmuistin tietorakennetta käydään turhaan läpi lineaarisesti.

Sovelluksen suorituskyvyn optimointi ei varsinaisesti kuulunut projektin laajuuteen, mutta nykyinen suorituskyky on mitä ilmeisimmin riittävä melko suurtenkin aliverkkomäärien käsittelyyn. Oman kokemukseni perusteella keskisuuren yrityksen aliverkkojen lukumäärä on yleensä muutamia kymmeniä tai korkeintaan muutamia satoja.

Go-ohjelmointiympäristön testausominaisuudet

Go-ohjelmointiympäristön testityökaluohjelmat ovat riittävän monipuoliset ohjelmistokehityksen tarpeisiin. Testausta varten Gossa on suoraan ominaisuudet yksikkö- ja kattavuustestaukseen (code coverage). Kattavuustestauksella tarkoitetaan tässä yhteydessä lausekattavuutta (statement coverage). Testiajo tuottaa teksti- tai HTML-muotoisen raportin, josta nähdään funktioiden ja metodien lausekattavuus prosentteina. (Donovan and Kernighan 2015: 318–320; Testing - The Go Programming Language 2016.)

`Testing`-ohjelmapaketti lisätään testiohjelmaan `import "testing"` -lausekkeella. Paketin avulla voidaan ohjelmoida testifunktioita, joiden nimien pitää alkaa `func Test` -merkkijonolla. Testikirjaston (test suite) yksittäisen lähdekoodimuotoisen testiohjelman nimen pitää päättyä merkkeihin `_test.go`, ja ohjelma sijoitetaan paketin tiedostohierarkian mukaiseen kansioon tai hakemistoon. Komento `go test` suorittaa automaattisesti kaikki sovelluspaketille testikirjastoon laaditut testifunktiot. Yksittäisen testifunktion nimen pitäisi olla mahdollisimman kuvaava, ja sen avulla pitäisi pystyä tunnistamaan testattava ominaisuus, metodi tai funktio. Testikirjaston testien suoritus joko onnistuu (pass) tai epäonnistuu (fail). Epäonnistuneista testeistä ruudulle tulostetaan lähdekooditiedoston nimi ja rivinumero, missä testausvirhe tapahtui. (Testing - The Go Programming Language 2016.)

Tavallisen yksikkötestauksen lisäksi `go test –bench` -komennolla voi suorittaa sovelluksen suorituskykyä mittaavia testejä (benchmark), joissa testifunktiota suoritetaan toistuvasti esimerkiksi 1 000 000 kertaa. Mittauksen tuloksena saadaan testin nimen ja suorituskertojen määrän lisäksi yksittäisen toiston suoritusaika nanosekunteina. (Testing - The Go Programming Language 2016.)

4 Sovelluksen esittely

Esimerkissä 1 esitellään kahden IPv4-verkon ja niiden aliverkkojen lisäämistä IP-suunnitelmaan. Verkot lisätään ensin sovelluksen ylimmän tason varantoon. Asiakkaalle luodaan uusi aliverkkosuunnitelma, johon varannon kaksi verkkoa lisätään. Toisen yksityisen IPv4-verkon peite on /24 ja toisen /16. Ensimmäinen verkko pilkotaan kahdeksaan aliverkkoon ja toinen 256 aliverkkoon. Lopuksi suunnitelman aliverkot tallennetaan JSON-muodossa ohjelmakansioon luotavaan tiedostoon.

```
Pool is empty
Pool loaded: Mar 5 2016 17:49:58 EET
> pool add 192.168.1.0/24        //verkon lisäys varantoon
> plan add 192.168.1.0/24        //verkon lisäys suunnitelmaan
> plan name TestClient           //suunnitelman nimeäminen
> show plan
Name: TestClient
192.168.1.0/24
> sub count 192.168.1.0/24 8     //pilkkominen kahdeksaan aliverkkoon
> show net 192.168.1.0/24
| 1-0 <nonf> 192.168.1.0/25
|| 2-0 <nonf> 192.168.1.0/26
||| 3-0 <free> 192.168.1.0/27
||| 3-20 <free> 192.168.1.32/27
|| 2-40 <nonf> 192.168.1.64/26
||| 3-40 <free> 192.168.1.64/27
||| 3-60 <free> 192.168.1.96/27
| 1-80 <nonf> 192.168.1.128/25
|| 2-80 <nonf> 192.168.1.128/26
||| 3-80 <free> 192.168.1.128/27
||| 3-a0 <free> 192.168.1.160/27
|| 2-c0 <nonf> 192.168.1.192/26
||| 3-c0 <free> 192.168.1.192/27
||| 3-e0 <free> 192.168.1.224/27
15 subnets
> show net 192.168.1.0/24 status free
||| 3-0 <free> 192.168.1.0/27
||| 3-20 <free> 192.168.1.32/27
||| 3-40 <free> 192.168.1.64/27
||| 3-60 <free> 192.168.1.96/27
||| 3-80 <free> 192.168.1.128/27
||| 3-a0 <free> 192.168.1.160/27
||| 3-c0 <free> 192.168.1.192/27
||| 3-e0 <free> 192.168.1.224/27
8 subnets
> net name 192.168.1.0/24 RootNet          //verkon nimeäminen
> net name 192.168.1.0/25 Data
> net name 192.168.1.32/27 LanHelsinki1
> net name 192.168.1.64/27 LanTurku1
> net name 192.168.1.96/27 LanPori1
> use 192.168.1.32/27                      //verkon käyttöönotto
> use 192.168.1.64/27
> use 192.168.1.128/27
```

```
> show net 192.168.1.0/24 status used
||| 3-20 <used> 192.168.1.32/27 LanHelsinki1
||| 3-40 <used> 192.168.1.64/27 LanTurku1
||| 3-60 <used> 192.168.1.96/27 LanPori1
3 subnets
> pool add 172.16.0.0/16                    //toisen verkon lisäys
> plan add 172.16.0.0/16
> net name 172.16.0.0/16 RootNet2
> sub prefix 172.16.0.0/16 /24
> show net 172.16.0.0/16 status free
|||||||| 8-0 <free> 172.16.0.0/24
|||||||| 8-100 <free> 172.16.1.0/24
|||||||| 8-200 <free> 172.16.2.0/24
|||||||| 8-300 <free> 172.16.3.0/24
|||||||| 8-400 <free> 172.16.4.0/24
|||||||| 8-500 <free> 172.16.5.0/24
|||||||| 8-600 <free> 172.16.6.0/24
|||||||| 8-700 <free> 172.16.7.0/24
. . .
|||||||| 8-fe00 <free> 172.16.254.0/24
|||||||| 8-ff00 <free> 172.16.255.0/24
256 subnets
> plan save                                 //suunnitelman tallennus
Name: TestClient
192.168.1.0/24
172.16.0.0/16
192.168.1.0/24 saved
192.168.1.0/25 saved
192.168.1.0/26 saved
192.168.1.0/27 saved
192.168.1.32/27 saved
192.168.1.64/26 saved
192.168.1.64/27 saved
. . .
```

Esimerkki 1. Kahden verkon lisäys, aliverkotus, nimeäminen ja tallennus.

Esimerkissä 2 sovelluksen varantoon lisätään uusi IPv6-verkko, joka lisätään myös asiakassuunnitelmaan. Asiakassuunnitelmalle annetaan kuvaava nimi. Lopuksi /48-peitteen IPv6-verkko nimetään ja pilkotaan /52-peitteen aliverkoiksi.

```
> pool add 2001:db8::/48                    //IPv6-verkon lisääminen
> plan add 2001:db8::/48
> plan name TestiClient2                    //Verkon nimeäminen
> sub prefix 2001:db8::/48 /52
> show net 2001:db8::/48 status free        //Vapaiden verkkojen lista
|||| 4-0 <free> 2001:db8::/52
|||| 4-1000 <free> 2001:db8:0:1000::/52
|||| 4-2000 <free> 2001:db8:0:2000::/52
|||| 4-3000 <free> 2001:db8:0:3000::/52
|||| 4-4000 <free> 2001:db8:0:4000::/52
|||| 4-5000 <free> 2001:db8:0:5000::/52
|||| 4-6000 <free> 2001:db8:0:6000::/52
|||| 4-7000 <free> 2001:db8:0:7000::/52
|||| 4-8000 <free> 2001:db8:0:8000::/52
|||| 4-9000 <free> 2001:db8:0:9000::/52
```

```
|||| 4-a000 <free> 2001:db8:0:a000::/52
|||| 4-b000 <free> 2001:db8:0:b000::/52
|||| 4-c000 <free> 2001:db8:0:c000::/52
|||| 4-d000 <free> 2001:db8:0:d000::/52
|||| 4-e000 <free> 2001:db8:0:e000::/52
|||| 4-f000 <free> 2001:db8:0:f000::/52
16 subnets
```

Esimerkki 2. IPv6-verkon lisäys suunnitelmaan ja pilkkominen aliverkkoihin.

5 Pohdinta

Insinöörityön tavoitteet, jotka olivat IPv4- ja IPv6-osoitealueiden suunnittelusovelluksen määrittely, suunnittelu, toteutus ja testaus, tutustuminen IPv4- ja IPv6-osoitesuunnittelun ja IPAM:n periaatteisiin sekä perehtyminen sovelluskehityksen käytäntöihin, saavutettiin suurelta osin. Projektin tuotoksena saatu komentorivipohjainen sovellusohjelma on toimiva ja käyttökelpoinen esiversio. Sovelluksen avulla voidaan tehdä useita verkkoja sisältävä suunnitelma, joka skaalautuu muutamista aliverkoista useisiin tuhansiin aliverkkoihin.

Projektin vaatimusmäärittelyn kahden ensimmäisen vaiheen vaatimuksista toteutettiin insinöörityöraportin palautuspäivään mennessä 97 prosenttia. Yksikkötestauksen osalta saavutettiin 82 prosentin kattavuus tärkeän IP-ohjelmakirjaston osalta.

Projektisuunnitelman alkuperäisistä tavoitteista jouduttiin matkan varrella kuitenkin tinkimään aikataulun kireyden, osin tietotaidon puutteiden ja etenkin projektin laajuuden määrittelyn ja aikataulusuunnittelun virheiden takia. Vaihejaon osalta graafinen käyttöliittymä jätettiin sovitusti suunnittelun ja toteutuksen ulkopuolelle noin 400. työtunnin kohdalla. Selainkäytön toteutus olisi lisännyt työmäärää tarpeettoman suureksi, ja projektia olisi pitänyt jatkaa kesällä 2016. Tavoiteaikataulun kireys oli hyvin tiedossa jo projektin asettamishetkellä, ja etenkin toive valmistumisesta kevään 2016 aikana ohjasi insinöörityön suunnittelua ja toteutusta. Tavoite insinöörityön ja -raportin valmistumisesta toteutui pääosin, vaikka aikataulussa ja projektin laajuudessa jouduttiinkin tekemään kompromisseja.

Projektisuunnitelman vaihejako olisi voinut olla toisenlainenkin. Alun perin ensimmäisen vaiheen tavoitteena oli riittävän laajan IP-ohjelmakirjaston toteutus. Sen avulla oli tarkoitus käsitellä ja muokata IP-osoitteita ja mahdollistaa myöhempien vaiheiden suju-

va toteutus. Ajatus oli sinänsä hyvä, mutta jälkeenpäin ajatellen oli hyvin vaikea ennakoida ja suunnitella etukäteen yksityiskohtaisesti, mitä funktioita, olioita tai metodeja tietyn toiminnon ohjelmointi vaatii ohjelmakirjastolta. Parempi lähestymistapa olisi ollut aloittaa tärkeistä toiminnallisista vaatimuksista suoraan ja toteuttaa samalla niihin liittyviä IP-ohjelmakirjaston osia alatehtävinä.

Toinen vaihejakoon liittynyt ongelma oli tietokantatallennuksen puuttuminen, mikä hankaloitti erityisesti testausta ja koekäyttöä, sillä kehitystyön alkupuolella sovelluksessa ei ollut minkäänlaista tallennusmahdollisuutta. Ongelma korjattiin ohjelmoimalla lisäfunktioita, jotka mahdollistavat IP-verkkojen tallennuksen teksti- ja JSON-muodossa. Vaatimusdokumentin ulkopuolisen muutoksen toteutus vei ajallisesti lähes puolitoista viikkoa.

Insinöörityön tuloksena saatua sovellusta ei voida luokitella varsinaiseksi IPAM-järjestelmäksi, koska siitä puuttuvat tärkeät DNS- ja DHCP-liitännät sekä IP-osoitteiden seurantaominaisuudet, mutta kyseessä ei ole yksinkertainen IP-laskurikaan. IP-laskureissa ei yleensä ole mahdollisuutta hierarkkiseen tai rekursiiviseen aliverkotukseen, ja monet niistä eivät tue IPv6-osoitteita lainkaan. Sovelluksen toimintoja on myös kohtalaisen helppo laajentaa tarvittaessa.

Lähteet

Aimonetti, Matt. 2015. Go Bootcamp. Verkkodokumentti. <www.golangbootcamp.com/book>. 15.4.2015. Luettu 24.11.2015.

Astah Professional Reference Manual. 2013. Verkkodokumentti. Change Vision, Inc. <astah.net/tutorials/Astah%20Professional%20Reference%20Manual.pdf>. Luettu 24.3.2016.

Command godoc. 2016. Verkkodokumentti. GoDoc. <godoc.org/golang.org/x/tools/cmd/godoc>. 8.3.2016. Luettu 26.2.2016.

DDI. 2016. Verkkodokumentti. Wikipedia. <en.wikipedia.org/wiki/DDI>. Luettu 14.2.2016.

Deepankar, Medhi & Karthikeyan, Ramasamy. 2007. Network Routing: Algorithms, Protocols, and Architectures. USA: Elsevier. Morgan Kaufmann Publishers.

Donovan, Alan. 2015. Go oracle: design. Verkkodokumentti. <golang.org/s/oracle-design>. 13.2.2015. Luettu 22.2.2016.

Donovan, Alan & Kernighan, Brian. 2015. The Go Programming Language. USA: Addison-Wesley.

Effective Go - The Go Programming Language. 2015. Verkkodokumentti. Golang.org. <golang.org/doc/effective_go.html>. Luettu 20.10.2015.

Fowler, Martin. 2003. Technical Debt. Verkkodokumentti. <martinfowler.com/bliki/TechnicalDebt.html>. 1.10.2003. Luettu 19.2.2016.

Fuller, Vince & Li, Tony. 2006. Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan. Verkkodokumentti. <tools.ietf.org/html/rfc4632>. 08.2006. Luettu 29.2.2016.

Gerrand, Andrew. 2011. Godoc: documenting Go code - The Go Blog. Golang.org. Verkkodokumentti. <blog.golang.org/godoc-documenting-go-code>. 31.3.2011. Luettu 26.2.2016.

GitHub codegangsta/cli. 2016. Verkkodokumentti. GitHub. <github.com/codegangsta/cli>. Luettu 4.3.2016.

Gocode - An autocompletion daemon for the Go programming language. 2016. Verkkodokumentti. GitHub. <github.com/nsf/gocode>. Päivitetty 18.1.2016. Luettu 21.2.2016.

Haikala, Ilkka & Mikkonen, Tommi. 2011. Ohjelmistotyön käytännöt. 12. painos. Talentum.

Hinden, Robert & Deering, Stephen. 2006. IP Version 6 Addressing Architecture. Verkkodokumentti. <tools.ietf.org/html/rfc4291>. 02.2006. Luettu 1.3.2016.

Hubbard, Patrick. 2012. Got IP address management problems? Think DNS, DHCP, IPAM trifecta. Verkkodokumentti. <searchnetworking.techtarget.com/feature/Got-IP-address-management-problems-Think-DNS-DHCP-IPAM-trifecta>. 11.2012. Luettu 16.2.2016.

Introduction to OMG's Unified Modeling Language® (UML®). 2015. Verkkodokumentti. Object Management Group, Inc. <www.omg.org/gettingstarted/what_is_uml.htm>. 24.6.2015. Luettu 17.2.2016.

IP address management. 2016. Verkkodokumentti. Wikipedia. <en.wikipedia.org/w/index.php?title=IP_address_management&oldid=708233248>. 4.3.2016. Luettu 7.3.2016.

IPv4 Address Allocation and Assignment Policies for the RIPE NCC Service Region. 2015. Verkkodokumentti. RIPE Network Coordination Centre. <www.ripe.net/publications/docs/ripe-649>. 07.2015. Luettu 24.2.2016.

Kawamura, Seiichi & Kawashima, Masanobu. 2010. A Recommendation for IPv6 Address Text Rep-resentation. Verkkodokumentti. <tools.ietf.org/html/rfc5952>. 08.2010. Luettu 14.2.2016.

Kerievsky, Joshua. 2005. Refactoring to Patterns. Boston, USA: Pearson Education. Addison-Wesley Signature Series.

Lerner, Andrew & Canales, Christian. 2015. Market Guide for DNS, DHCP and IP Address Management (DDI). Verkkodokumentti. <www.gartner.com/doc/reprints?id=1-2AISCZT&ct=150225&st=sb>. 24.2.2015. Luettu 14.2.2016.

Medeiros, Bruno. 2016. GitHub - GoClipse/goclipse: Eclipse IDE for the Go programming language. Verkkodokumentti. <github.com/GoClipse/goclipse>. Päivitetty 4.3.2016. Luettu 21.2.2016.

Narten, Thomas & Huston, Geoff. 2011. IPv6 Address Assignment to End Sites. Verkkodokumentti. <tools.ietf.org/html/rfc6177>. 03.2011. Luettu 14.2.2016.

Nathalie, Nathalie. 2016. 10,000 LIRs with IPv6 Resources — RIPE Labs. Verkkodokumentti. <labs.ripe.net/Members/nathalie_nathalie/10-000-lirs-with-ipv6-resources>. 29.2.2016. Luettu 3.1.2016.

Preparing an IPv6 Address Plan Manual. 2013. Verkkodokumentti. SURFnet. <www.ripe.net/support/training/material/IPv6-for-LIRs-Training-Course/Preparing-an-IPv6-Addressing-Plan.pdf>. 18.9.2013. Luettu 15.2.2016.

RIPE NCC Statistics. 2016. Verkkodokumentti. RIPE Network Coordination Centre. <labs.ripe.net/statistics/?tags=ipv6>. Luettu 27.2.2016.

Rooney, Timothy. 2011. IP Address Management: Principles and Practice, IEEE Press Series on Network Management. Hoboken, New Jersey, USA: John Wiley & Sons.

Rooney, Timothy. 2013a. IP Address Management (IPAM) Best Practices. Verkkodokumentti. Cisco Systems. <www.cisco.com/c/en/us/products/collateral/cloud-systems-management/prime-network-registrar/white_paper_c11-728639.pdf>. 09.2013. Luettu 14.2.2016.

Rooney, Timothy. 2013b. IPv6 Address Planning: Guidelines for IPv6 address allocation. Verkkodokumentti. The Internet Society Deploy360 Programme. <www.internetsociety.org/deploy360/resources/ipv6-address-planning-guidelines-for-ipv6-address-allocation/>. 24.9.2013. Luettu 8.3.2016.

Testing - The Go Programming Language. 2016. Verkkodokumentti. Golang.org. <golang.org/pkg/testing/>. Luettu 22.2.2016.

The Go Programming Language Specification. 2015. Verkkodokumentti. Golang.org. <golang.org/ref/spec>. 5.1.2016. Luettu 16.2.2016.

Thurston, Adrian. 2009. Ragel State Machine Compiler - User Guide. Verkkodokumentti. <www.colm.net/files/ragel/ragel-guide-6.4.pdf>. 03.2009. Luettu 29.10.2015.

Tkachenko, Andrey. 2015. Go for C++ Programmers. Verkkodokumentti. GitHub. <github.com/golang/go/wiki/GoForCPPProgrammers>. 7.12.2015. Luettu 29.2.2016.

Usage Statistics of IPv6 for Websites. 2016. Verkkodokumentti. W3techs. <w3techs.com/technologies/details/ce-ipv6/all/all>. 14.2.2016. Luettu 14.2.2016.

Vorgehensmodell V-Modell: T.3 IT-Vorhabentypen 2004. Verkkodokumentti. Universität Bremen. <www.informatik.uni-bremen.de/uniform/gdpa/part3_d/p3t2.htm>. 3.3.2004. Luettu 18.3.2016.